

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e Automação Industrial

Plano de Segurança para Autenticação de Dados em Redes Orientadas à Informação

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do **título de Doutor em Engenharia Elétrica**. Área de concentração: Engenharia de Computação.

Autor: Walter Wong
Orientador: Maurício Ferreira Magalhães
Co-orientador: Jussi Kangasharju

Campinas, SP
2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

W846p Wong, Walter
 Plano de segurança para autenticação de dados em
 redes orientadas à informação / Walter Wong. – Campinas,
 SP: [s.n.], 2011.

Orientadores: Maurício Ferreira Magalhães; Jussi
Kangasharju.

Tese de Doutorado - Universidade Estadual de
Campinas, Faculdade de Engenharia Elétrica e de
Computação.

1. Arquitetura de redes de computadores. 2. Internet.
3. Computadores - Controle de acesso. 4. Redes de
computação - Protocolos. I. Magalhães, Maurício
Ferreira. II. Kangasharju, Jussi. III. Universidade
Estadual de Campinas. Faculdade de Engenharia Elétrica e
de Computação. IV. Título

Título em Inglês:	Security plane for data authentication in information-centric networks
Palavras-chave em Inglês:	Architecture of computer networks, Internet, Computers - Access control, Computer Networks - Protocols
Área de concentração:	Engenharia de Computação
Titulação:	Doutor em Engenharia Elétrica
Banca Examinadora:	Luis Fernando Faina, Marcos Rogério Salvador, Ricardo Dahab, Marco Aurélio Amaral Henriques
Data da defesa:	23-09-2011
Programa de Pós Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Walter Wong

Data da Defesa: 23 de setembro de 2011

Título da Tese: "Plano de Segurança para Autenticação de Dados em Redes Orientadas à Informação"

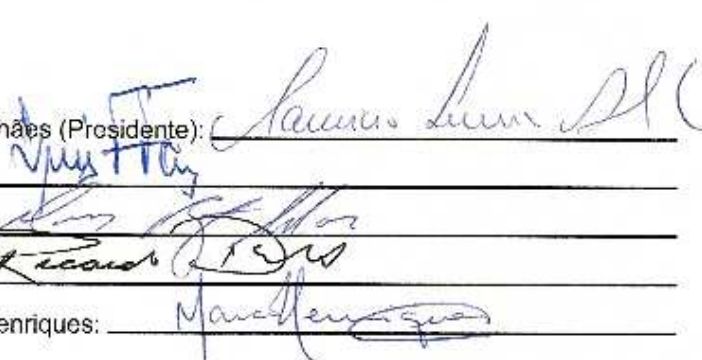
Prof. Dr. Maurício Ferreira Magalhães (Presidente):

Prof. Dr. Luis Fernando Faina:

Dr. Marcos Rogério Salvador:

Prof. Dr. Ricardo Dahab:

Prof. Dr. Marco Aurélio Amaral Henriques:

The image shows five handwritten signatures in blue ink, each written over a horizontal line. The signatures are: 1. Prof. Dr. Maurício Ferreira Magalhães (Presidente): A large, stylized signature. 2. Prof. Dr. Luis Fernando Faina: A signature that appears to be 'Luis Faina'. 3. Dr. Marcos Rogério Salvador: A signature that appears to be 'Marcos Salvador'. 4. Prof. Dr. Ricardo Dahab: A signature that appears to be 'Ricardo Dahab'. 5. Prof. Dr. Marco Aurélio Amaral Henriques: A signature that appears to be 'Marco Aurélio Amaral Henriques'.

Resumo

A segurança da informação é responsável pela proteção das informações contra o acesso não-autorizado, uso, modificação ou a sua destruição. Com o objetivo de proteger os dados contra esses ataques de segurança, vários protocolos foram desenvolvidos, tais como o *Internet Protocol Security* (IPSEC) e o *Transport Layer Security* (TLS), provendo mecanismos de autenticação, integridade e confidencialidade dos dados para os usuários. Esses protocolos utilizam o endereço IP como identificador de *hosts* na Internet, tornando-o referência e identificador no estabelecimento de conexões seguras para a troca de dados entre aplicações na rede.

Com o advento da Web e o aumento exponencial do consumo de conteúdos, como vídeos e áudios, há indícios da migração gradual do uso predominante da Internet, passando da ênfase voltada para a conexão entre *hosts* para uma ênfase voltada para a obtenção de conteúdo da rede, paradigma esse conhecido como *information-centric networking*. Nesse paradigma, usuários buscam por documentos e recursos na Internet sem se importarem com o conhecimento explícito da localização do conteúdo. Como consequência, o endereço IP que previamente era utilizado como ponto de referência do provedor de dados, torna-se meramente um identificador efêmero do local onde o conteúdo está armazenado, resultando em implicações para a autenticação correta dos dados. Nesse contexto, a simples autenticação de um endereço IP não garante a autenticidade dos dados, uma vez que o servidor identificado por um dado endereço IP não é necessariamente o endereço do produtor do conteúdo.

No contexto de redes orientadas à informação, existem propostas na literatura que possibilitam a autenticação dos dados utilizando somente o conteúdo propriamente dito, como a utilização de assinaturas digitais por bloco de dado e a construção de árvores de *hash* sobre os blocos de dados. A ideia principal dessas abordagens é atrelar uma informação do provedor original do conteúdo nos blocos de dados transportados, por exemplo, uma assinatura digital, possibilitando a autenticação direta dos dados com o provedor, independentemente do *host* onde o dado foi obtido. Apesar do mecanismo citado anteriormente possibilitar tal verificação, esse procedimento é muito oneroso do ponto de vista de processamento, especialmente quando o número de blocos é grande, tornando-o inviável de ser utilizado na prática.

Este trabalho propõe um novo mecanismo de autenticação utilizando árvores de *hash* com o objetivo de prover a autenticação dos dados de forma eficiente e explícita com o provedor original e, também, de forma independente do *host* onde os dados foram obtidos. Nesta tese, propomos duas técnicas de autenticação de dados baseadas em árvores de *hash*, chamadas de *skewed hash tree* (SHT) e *composite hash tree* (CHT), para a autenticação de dados em redes orientadas à informação. Uma vez criadas, parte dos dados de autenticação é armazenada em um *plano de segurança* e uma outra parte permanece acoplada ao dado propriamente dito, possibilitando a verificação baseada no conteúdo e não no *host* de origem. Além disso, essa tese apresenta o modelo formal, a especificação e a implementação das duas técnicas de árvore de *hash* para autenticação dos dados em redes de conteúdo através de um plano de segurança. Por fim, esta tese detalha a instanciação do modelo de plano de segurança proposto em dois cenários de autenticação de dados: 1) redes *Peer-to-Peer* e 2) autenticação paralela de dados sobre o HTTP.

Palavras-chave: Segurança, arquiteturas de nova geração, protocolos de rede.

Abstract

Information security is responsible for protecting information against unauthorized access, use, modification or destruction. In order to protect such data against security attacks, many security protocols have been developed, for example, Internet Protocol Security (IPSec) and Transport Layer Security (TLS), providing mechanisms for data authentication, integrity and confidentiality for users. These protocols use the IP address as host identifier on the Internet, making it as a reference and identifier during the establishment of secure connections for data exchange between applications on the network.

With the advent of the Web and the exponential increase in content consumption (e.g., video and audio), there is an evidence of a gradual migration of the predominant usage of the Internet, moving the emphasis on the connection between hosts to the content retrieval from the network, which paradigm is known as information-centric networking. In this paradigm, users look for documents and resources on the Internet without caring about the explicit knowledge of the location of the content. As a result, the IP address that was used previously as a reference point of a data provider, becomes merely an ephemeral identifier of where the content is stored, resulting in implications for the correct authentication data. In this context, the simple authentication of an IP address does not guarantee the authenticity of the data, because a hosting server identified by a given IP address is not necessarily the same one that is producing the requested content.

In the context of information-oriented networks, some proposals in the literature proposes authentication mechanisms based on the content itself, for example, digital signatures over a data block or the usage of hash trees over data blocks. The main idea of these approaches is to add some information from the original provider in the transported data blocks, for example, a digital signature, enabling data authentication directly with the original provider, regardless of the host where the data was obtained. Although the mechanism mentioned previously allows for such verification, this procedure is very costly in terms of processing, especially when the number of blocks is large, making it unfeasible in practice.

This thesis proposes a new authentication mechanism using hash trees in order to provide efficient data authentication and explicitly with the original provider, and also independently of the host where the data were obtained. We propose two techniques for data authentication based on hash trees, called *skewed hash tree* (SHT) and *composite hash tree* (CHT), for data authentication in information-oriented networks. Once created, part of the authentication data is stored in a *security plane* and another part remains attached to the data itself, allowing for the verification based on content and not on the source host. In addition, this thesis presents the formal model, specification and implementation of two hash tree techniques for data authentication in information-centric networks through a security plane. Finally, this thesis details the instantiation of the security plane model in two scenarios of data authentication: 1) Peer-to-Peer and 2) parallel data authentication over HTTP.

Keywords: Security, clean slate architecture, network protocols.

Agradecimentos

To my wife Amy Yuan Shao, who has blindly supported me ever since in my endeavor towards my Ph.D. Without her support, specially abroad, this work would not be possible.

To my advisor, prof. Maurício Ferreira Magalhães, I am thankful for his support and advising since 2003, starting with the scientific initiation program until this present moment. During these many years, I have learned many aspects about academic research, which would not be possible without his advising.

To my supervisor, prof. Jussi Kangasharju, who has trusted and hosted me at the University of Helsinki between 2009-2010. Without his warm discussions and insights, I would not be able to learn new views about my research.

To Pekka Nikander, who has believed on me in 2008 and made possible my trip to Finland in the following year. Without his support, I would not have had the opportunity to interact with many other researchers from different fields and learn about corporate research.

To my colleagues and friends in the laboratory, which I am grateful for the sugestions, discussions and specially for the spare time that made my life much more interesting there.

To CAPES and Ericsson Research for their financial support.

To my beloved wife Amy Yuan Shao who always supported me.

Sumário

Lista de Figuras	xiii
Lista de Tabelas	xv
Glossário	xvii
Trabalhos Publicados Pelo Autor	xxi
1 Introdução	1
1.1 Contribuições do Trabalho	3
1.2 Text Organization	11
2 Related Work	13
2.1 Authentication Schemes	13
2.1.1 Lamport-Diffie One-time Signature (1979)	13
2.1.2 Merkle Hash Tree (1989)	15
2.2 Standard Security Protocols	21
2.2.1 Secure Sockets Layer (1994)	22
2.2.2 Internet Protocol Security (1998)	23
2.3 New Security Protocols	24
2.3.1 Packet Level Authentication (2005)	24
2.3.2 Host Identity Protocol (2006)	24
2.3.3 An adaptive and lightweight protocol for hop-by-hop authentication (2008)	25
2.4 Clean Slate Internet Architectures	25
2.4.1 A Data-oriented (and Beyond) Network Architecture (2007)	25
2.4.2 Publish-Subscribe Internet Routing Paradigm (2008)	26
2.4.3 4WARD Project (2008)	27
2.4.4 Content-centric Networking (2009)	27
2.5 Summary	28
3 Towards Identity-centric Security	29
3.1 Motivation	29
3.2 Identification Layer	30
3.3 Identification Layer Security Model	32
3.3.1 Security Model Proposal	32

3.3.2	Protocol Implementation	36
3.3.3	Experimental Evaluation	37
3.4	Summary	40
4	Data Authentication in Information-centric Networks	43
4.1	Motivation	43
4.2	Towards Information-centric Networking	45
4.2.1	Security Model	45
4.3	Information-centric Authentication Mechanisms	47
4.3.1	Skewed Hash Tree	47
4.3.2	Composite Hash Tree	54
4.4	Summary	64
5	Application Scenarios for Information-centric Data Authentication	65
5.1	Towards Secure Name Resolution	65
5.1.1	Motivation	65
5.1.2	Naming System Design	66
5.1.3	Implementation architecture	67
5.1.4	Security Analysis	71
5.1.5	Deployment Considerations	71
5.1.6	Summary	72
5.2	Peer-to-peer Pollution Detection	73
5.2.1	Background	73
5.2.2	Attacks against P2P Systems	75
5.2.3	Piece Fingerprinting Design	77
5.2.4	Application in BitTorrent	80
5.2.5	Summary	86
5.3	Parallel Authentication over HTTP	87
5.3.1	Background	87
5.3.2	Parallel Verification Proposal	90
5.3.3	Evaluation	91
5.3.4	Related Approaches	95
5.3.5	Summary	97
5.4	Secure Content Caching	98
5.4.1	Network Caching Design	99
5.4.2	Implementation	103
5.4.3	Evaluation	104
5.4.4	Experimental Results & Analysis	105
5.4.5	Summary	107
5.5	Summary	109
6	Conclusions	111
6.1	Future Work	112

Lista de Figuras

2.1	OTSS key generation phases. (a) Private key generation using a Pseudo-random Number generator. (b) Public key generation from the private key.	14
2.2	OTSS signature generation using the private key.	14
2.3	OTSS signature verification procedure.	15
2.4	(a) Merkle Tree of height $H = 2$ with $2^H = 4$ leaves. (b) Data blocks with their corresponding <i>Authentication Paths</i>	16
2.5	Authentication of data block zero using Merkle Trees.	17
2.6	Unbalanced Merkle Tree ($H = 4$) with naive zero padding scheme.	18
2.7	(a) Regular Merkle Tree construction. (b) <i>Authentication paths</i> associated to each data block. The repeated hash values are highlighted in red.	20
2.8	Percentage of repeated hash values in the AP versus the height of Merkle Tree. . . .	21
2.9	(a) TLS between the transport and application layer. (b) Multiple secure channels between the same end-hosts.	22
2.10	(a) IPSEC transport mode. (b) IPSEC tunnel mode.	23
3.1	(a) Standard TCP/IP protocol stack. (b) Protocol stack with the identification layer. .	31
3.2	Security association establishment protocol.	33
3.3	Cost function example used to increase the costs of a denial-of-service attack. . . .	34
3.4	New Internet architecture framework.	36
3.5	Identification Layer Security Header.	38
3.6	Mobility evaluation scenarios. In (1), node B is attached to the access point 1 to communicate with node A. In (2), node B moves and uses the access point 2 to communicate with node A.	40
4.1	Security plane overview.	45
4.2	Metadata description.	46
4.3	Skewed Merkle Tree with 6 data blocks and one appended level in the skewed hash tree with height 2.	48
4.4	(a) Skewed Merkle Tree with 11 leaves and the inner balanced tree with 8 leaves. (b) <i>Authentication Path</i> for each data block with different lengths.	49
4.5	(a) Skewed hash tree with $h = 2$. (b) Table containing the output on each round of the algorithm.	53
4.6	(a) Composite Hash Tree with internal HT of height $h = 1$ and $\alpha = 2$	56
4.7	Authentication Window for CHT.	57

4.8	CHT(4, 1) for 16 data blocks with $H = 2$	58
5.1	URI naming scheme and resolution in the current DNS.	68
5.2	Naming scheme and resolution to authority and metadata.	69
5.3	Trust establishment and transfer from the names to the data blocks.	69
5.4	Metadata fields used in the name resolution.	70
5.5	Skewed hash tree applied over a set of data blocks.	70
5.6	Example of file fragmentation in P2P networks. The file is divided into a set of pieces and each piece is divided into a set data chunks.	75
5.7	An example of total pollution attack in P2P networks.	77
5.8	An example of partial pollution attack in P2P networks.	78
5.9	(a) Example of a CHT($\alpha = 2, h = 1$). (b) Data chunks with their respective FVL. . .	79
5.10	Data chunk authentication hierarchy for CHT(1, 2).	80
5.11	A BitTorrent metadata file (.torrent) with a list of <i>root fingerprints</i>	81
5.12	Integrity verification of data blocks using a fingerprint list.	82
5.13	(a) CHT ($\alpha = 1400, h = 4$). (b) Each data block has four verification fingerprints and the resulting <i>Root Fingerprint</i> of all blocks should be ID_1 to pass the integrity check.	84
5.14	Bandwidth consumption versus the number of polluted blocks within each piece using CHT in BitTorrent.	85
5.15	Parallel data verification scenario. (a) First, the application retrieves the RH and verifies the digital signature. (b) The application retrieves ADs and subsequent data blocks from multiple sources. (c) Data blocks are verified using the previously received ADs.	91
5.16	Example of authentication hierarchy for CHT(1, 2).	92
5.17	CHT Overhead comparison using different internal hash trees for a file of 1GB divided in blocks of 32, 64, 128, 256, 512KB.	94
5.18	Hierarchy dependency vs. aggregation index (α) using different internal hash tree heights.	94
5.19	Legacy application support using cross-application. (a) Applications can <i>opportunistically</i> retrieve data blocks from different sources. (b) Applications retrieve the <i>Authentication Paths</i> from an authentication server.	96
5.20	Content routers with internal neighborhood table.	100
5.21	A content router with its neighbor zones used for the chunk discovery.	101
5.22	General view of the secure caching mechanism.	102
5.23	Internal modules of a content router.	103
5.24	Caching control header	104
5.25	Evaluation topology with one CR.	105
5.26	Skewed Hash Tree Root Hash Generation Times.	106
5.27	Skewed Hash Tree AP generation times and 1024-bit RSA signature times comparison.	106
5.28	Skewed Hash Tree AP verification times and 1024-bit RSA signature verification times comparison.	107

Lista de Tabelas

2.1	Merkle Tree height vs. number of data blocks	19
3.1	Security model overhead vs. IDSEC modes (95% Confidence Interval)	39
3.2	Handover time vs. IDSEC security modes	40
4.1	Merkle Tree vs. Composite Hash Tree Overhead Comparison	60
5.1	Usual BitTorrent piece size	76
5.2	CHT authentication overhead vs. required authentication hierarchies	84
5.3	CHT overhead vs. authentication hierarchies	93
5.4	CHT overhead (MB) vs. file size using data chunks of 64 KB.	93
5.5	Comparison between verification techniques	96
5.6	Signature and verification speeds with different cryptographic algorithms	105

Glossário

ACK - Acknowledgement

AD - Authentication Data Block

ADHT - Authenticated Distributed Hash Table

AH - Authentication Header

ALPHA - Adaptive and Lightweight Protocol for Hop-by-hop Authentication

AP - Authentication Path

API - Application Programming Interface

BO - Bit-level Object

CCN - Content-centric Networking

CDN - Content Delivery Networks

CH - Composite Root Hash

CHT - Composite Hash Tree

CR - Content Router

CVL - Chunk Verification List

DH - Diffie-Hellman

DHCP - Dynamic Host Configuration Protocol

DNS - Domain Name System

DNSSEC - Domain Name System Security Extensions

DO - Data Object

DONA - Data-oriented and Networking Architecture

DoS - Denial of Service

ESP - Encapsulating Security Payload

FL - Fingerprint List

FP - Fingerprint

FQDN - Fully Qualified Domain Name

FTP - File Transfer Protocol

HI - Host Identity

HIP - Host Identity Protocol

HIT - Host Identity Tag

HMAC - Hash-based Message Authentication Code

HT - Hash Tree

HTML - HyperText Markup Language

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

ICMP - Internet Control Message Protocol

ICN - Information-centric Networking

IDSEC - Identification Layer Security

IO - Information Object

IP - Internet Protocol

IPSEC - Internet Protocol Security

ISP - Internet Service Provider

MD5 - Message-Digest algorithm 5

MT - Merkle Tree

NAT - Network Address Translation

NDN - Named Data Network

NIC - Network Interface Controller

OTSS - One-Time Signature Scheme

P2P - Peer-to-peer

PFS - Perfect Forward Secrecy

PKI - Public Key Cryptography

PLA - Packet Level Authentication

PRNG - Pseudo Random Number Generator

PSIRP - Publish/Subscribe Internet Routing Paradigm

RF - Root Fingerprint

RH - Root Hash

RId - Rendezvous Identifier

RVS - Rendezvous Server

SA - Security Association

SAD - Security Association Database

SHA-1 - Secure Hashing Algorithm

SHT - Skewed Hash Tree

SId - Scope Identifier

SNP - Secure Network Programming

SPD - Security Policy Database

SSL - Secure Sockets Layer

TCP - Transmission Control Protocol

TLI - Transport Level Identification

TLS - Transport Layer Security

URI - Universal Resource Identifier

URL - Universal Resource Locator

VPN - Virtual Private Network

WWW - World Wide Web

Trabalhos Publicados Pelo Autor

1. Walter Wong, Mauricio Magalhães. Security Approaches for Information-centric Networks. Accepted for InTech Cryptography/Book 2. ISBN 979-953-307-863-1. 2011.
2. Walter Wong, Mauricio Magalhães. A Security Plane for Information-centric Networks. Submitted to Journal of Computer and Security (Elsevier).
3. Walter Wong, Liang Wang, Jussi Kangasharju. "INCA: In-network Cooperative Caching Architecture". Submitted to *IEEE INFOCOM'11*.
4. Walter Wong, Mauricio Magalhães. Skewed Hash Trees for Efficient Content Authentication. Submitted to IEEE Communication Letters.
5. Augusto Morales, Oscar Novo, Walter Wong, Tomas Valladares. Publish/Subscribe Communication Mechanisms over PSIRP. *7th IEEE International Conference on Next Generation Web Services Practices (NWeSP'11)*, Salamanca, Spain. October 19-21, 2011.
6. Walter Wong, Marcus Giraldi, Mauricio Magalhães, Jussi Kangasharju. "Content Routers: Fetching Data on Network Path". *IEEE International Conference on Communications (ICC'11)*, Kyoto, Japan. June 5-9, 2011.
7. Walter Wong, Pekka Nikander. "Towards Secure Information-centric Naming". *Securing and Trusting Internet Names Workshop (SATIN'11)*. Teddington, UK. April 4-5, 2011.
8. Walter Wong and Pekka Nikander. "Secure Naming in Information-centric Networks". *3rd ACM Re-Architecting the Internet (ReArch'10) Workshop, co-allocated with 6th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT'10)*. Philadelphia, USA. November 30 - December 3, 2010.
9. Walter Wong, Mauricio Magalhães, Jussi Kangasharju. "Piece Fingerprinting: Binding Content and Data Blocks Together in Peer-to-peer Networks". *IEEE Global Communications Conference (Globecom'10)*, Miami, Florida, USA. December 6-10, 2010.
10. Walter Wong, Mauricio Magalhães, Jussi Kangasharju. "Towards Verifiable Parallel Content Retrieval". *6th IEEE Workshop on Secure Network Protocols (NPSec'10), co-allocated with 18th International Conference on Network Protocols (ICNP'10)*, Kyoto, Japan, 2010. October 5-8, 2010.
11. Somaya Arianfar, Jorg Ott, Lars Eggert, Pekka Nikander, Walter Wong. "A Transport Protocol for Content-centric Networks". Extended Abstract. *18th International Conference on Network Protocols (ICNP'10)*, Kyoto, Japan, 2010. October 5-8, 2010.
12. Walter Wong, Marcus Giraldi, Fabio Verdi, Mauricio Magalhães. An Identifier-based Architecture for Native Vertical Handover Support. *24th International Conference on Advanced Information Networking and Applications (AINA'10)*, Perth, Australia. April 20-23, 2010.

13. Walter Wong, Fabio Verdi and Mauricio Magalhães. "IDSec: an Identification Layer Security Model". *24th International Conference on Advanced Information Networking and Applications (AINA'10)*, Perth, Australia. April 20-23, 2010.
14. Walter Wong, Fabio Verdi and Mauricio Magalhães. "A Security Plane for Publish/Subscribe based Content-oriented Networks". *4th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT'08) - Student Workshop*, Madrid, Spain. December 09-12, 2008.

Capítulo 1

Introdução

A arquitetura original da Internet foi concebida com o objetivo de interligar computadores geograficamente distantes para o compartilhamento de recursos e serviços, por exemplo, login remoto e transferência de arquivos. A arquitetura foi construída sobre a tecnologia de comutação de pacotes com o objetivo de aumentar a robustez da rede de comunicação, por exemplo, para ser robusta contra catástrofes naturais ou mesmo um cenário de guerra. O modelo básico de comunicação utiliza o Internet Protocol (IP) para troca de pacotes entre dois nós finais, onde a cada nó é atribuído um endereço IP único e permanente. Sempre que um nó deseja enviar um pacote, ele cria um pacote com o endereço IP de destino e roteadores ao longo do caminho realizam o roteamento até o destino desejado. Por outro lado, o destino pode enviar o pacote de volta para o nó de origem apenas invertendo o endereço IP de origem e de destino no cabeçalho IP, conforme o princípio da comunicação fim-a-fim [1].

O advento da World Wide Web (WWW) no início dos anos 90 fez com que a Internet se tornasse popular fora da comunidade acadêmica, tornando a obtenção de documentos e serviços, tais como páginas HTML e o comércio eletrônico, de fácil acesso aos usuários através de navegadores e servidores Web. Concorrentemente, novos requisitos de segurança surgiram como a autenticação de aplicativos e confidencialidade dos dados, especialmente com o surgimento de serviços bancários *online*. O esforço para prover mecanismos de segurança para aplicativos iniciou-se com o *secure network programming* (SNP) [2] e o *secure socket layer* (SSL) [3], que eram parte integrante do navegador Web da Netscape. O SSL evoluiria para um padrão conhecido como Transport Layer Security (TLS) [4], que está na versão 1.2 no momento desta tese. Além disso, o Internet Security (IPSEC) [4] foi proposto para fornecer segurança entre nós da rede, um modo de operação que o SSL não oferecia. O IPSEC oferece segurança na camada de rede, tais como autenticação, integridade e confidencialidade dos dados trocados entre dois nós. O TLS também provê mecanismos de segurança similares ao IPSEC, porém ele opera na camada de transporte.

A introdução de novos cenários de uso, tais como as redes sem fio e redes orientadas à informação, expôs algumas limitações da arquitetura da Internet atual. Uma dessas limitações é conhecida como o problema da sobrecarga semântica do IP [5, 6, 7], que consiste no uso simultâneo do endereço IP em ambas as camadas de rede e de transporte. Na camada de rede, o endereço IP é usado como um identificador topológico, indicando um ponto geográfico na topologia da Internet. Na camada de transporte, o endereço IP é usado como um identificador de nó, sendo utilizado como identificador de nó final durante o estabelecimento de uma conexão entre dois nós¹. Portanto, há um acoplamento

¹Para estabelecer uma conexão utilizando a API de *sockets* de Berkeley, uma aplicação precisa de um *transport level*

implícito entre os conceitos de identificação e localização², onde a identificação de um nó é vinculada a uma localização dentro de uma rede, impedindo cenários de mobilidade nativa sem modificações no identificador de nó final. Do ponto de vista de segurança, a ausência de um identificador permanente e seguro durante um evento de mobilidade do nó resulta em possíveis ataques de segurança, como a impersonificação e ataques *man-in-the-middle* [8].

Uma segunda limitação resultante da evolução da Internet se refere ao conflito gerado entre o uso atual da Internet e seu design original [9]. Atualmente, a Internet é predominantemente utilizada para a obtenção de conteúdo, como músicas e vídeos, independentemente do seu local de armazenamento na rede³. Por exemplo, os usuários acessam a Internet para obter informações relevantes, tais como previsões meteorológicas, notícias, etc, mas eles não estão interessados no endereço IP de onde o conteúdo está sendo obtido. Na verdade, os usuários estão mais interessados na procedência⁴ do conteúdo em vez do local onde o conteúdo foi obtido. Estas redes são conhecidas como *redes orientadas à informação* [10] ou, como cunhado por Van Jacobson, *redes orientados a conteúdo* e são divididas em dois tipos básicos: arquiteturas de nova geração e aplicações legadas orientadas à informação. O primeiro tipo representa novas abordagens de arquiteturas de nova geração que suportam por completo as funcionalidades das redes orientadas à informação, como as primitivas *publish/subscribe*[10] e *get/put*[11] ao invés das primitivas tradicionais *send/receive*. O segundo tipo representa os aplicativos que já estão implantados na arquitetura atual da Internet e são conceitualmente orientados à informação, porém são implementados sobre o paradigma *send/receive*. Alguns exemplos deste grupo incluem redes de distribuição de conteúdo (CDN) [12] e redes *Peer-to-peer* (P2P)[13]. O foco principal dessas redes é obter o conteúdo de forma eficiente sem o conhecimento prévio da fonte do conteúdo. Por exemplo, as redes CDN realizam a resolução de nomes através do uso de DNS dinâmico para o servidor de conteúdo mais próximo, enquanto as redes P2P realizam a resolução de nomes para a localização do *peer* provedor do conteúdo através de consultas a uma *distributed hash table*[14].

De acordo com [15], as principais características das redes orientadas a conteúdo são o desacoplamento espacial e temporal. O desacoplamento espacial refere-se ao desconhecimento prévio do endereço do servidor que hospeda o conteúdo antes da conexão. O desacoplamento temporal refere-se à ausência de uma sessão explícita entre clientes e servidores para se obter um bloco de conteúdo, por exemplo, entre a geração e o consumo de um conteúdo como as notícias. Por outro lado, a arquitetura original da Internet é orientada à conexão entre os nós finais, onde o conhecimento da localização de um nó na rede é um pré-requisito para a obtenção dos dados de forma síncrona. Do ponto de vista de segurança, a ausência de um identificador estável e uma sessão explícita entre dois nós representam um desafio arquitetural e técnico para a segurança do conteúdo nestes novos cenários de uso.

Os protocolos de segurança em uso atualmente (por exemplo, IPSEC e TLS) são síncronos e usam o endereço IP como identificador de nó durante o estabelecimento de uma conexão segura entre dois nós finais. No entanto, nas redes orientadas à informação, não há conhecimento prévio da localização da origem dos dados, levando a problemas de confiança da fonte. Por exemplo, a fonte de dados

identifier (TLI), que é composto de um par de endereços e portas de origem e de destino e a versão do protocolo de rede.

²O termo *localização* é utilizado como o ponto de ligação do nó à rede, representado pelo seu endereço IP na arquitetura da Internet.

³O termo *armazenamento* é utilizado com o sentido de localização na rede, ou seja, o endereço IP onde o servidor está localizado.

⁴A origem dos dados do produtor, por exemplo, se um bloco de dados são originados de uma origem confiável.

pode ser um *Web cache* ou um outro nó, mas não há nenhum mecanismo explícito para autenticar os nós provedores de conteúdo. Neste cenário, os protocolos de segurança podem fornecer mecanismos para a verificação da integridade do conteúdo em redes orientadas à informação, porém não provêm a validação. A verificação fornece mecanismos para verificar se um determinado conjunto de dados não foi alterado durante a transferência entre dois nós finais, enquanto a validação significa que o conteúdo recuperado é o que o cliente havia solicitado originalmente. Por exemplo, se um usuário busca por um conteúdo em uma rede P2P, mesmo que seja possível estabelecer uma conexão usando IPSEC ou TLS, o usuário somente poderá verificar que o conteúdo foi obtido de um determinado *peer* e que não foi modificado durante a transferência. No entanto, não há nenhum mecanismo para validar se o conteúdo obtido é realmente o conteúdo que o usuário havia originalmente solicitado.

Esta tese propõe um novo mecanismo de autenticação usando árvores de *hash*, a fim de prover autenticação eficiente dos dados e de forma explícita com o provedor original⁵, e também de forma independente do nó de onde os dados foram obtidos. Propomos duas técnicas para a autenticação de dados baseadas em árvores de *hash*, chamada de *skewed hash tree* (SHT) e *composite hash tree* (CHT), para a autenticação de dados em redes orientadas à informação. Uma vez criada, parte dos dados de autenticação é armazenada em um *plano de segurança* e a outra parte permanece junto com os dados, possibilitando a verificação com base no conteúdo e não no nó de origem. Além disso, esta tese apresenta o modelo formal, a especificação e a implementação das duas técnicas de árvore *hash* para autenticação dos dados em redes orientadas à informação através do plano de segurança. Finalmente, o trabalho propõe e detalha a instanciação do modelo do plano de segurança em dois cenários de autenticação de dados: 1) *Peer-to-peer* e 2) autenticação paralela de dados sobre HTTP.

1.1 Contribuições do Trabalho

A contribuição principal desta tese é a proposta de um modelo de segurança para a autenticação de dados em redes orientadas à informação, atendendo aos requisitos de desacoplamento espacial e temporal. Iniciamos a proposta com um modelo de segurança visando prover o desacoplamento espacial nos protocolos de segurança e, em seguida, generalizamos o modelo proposto para suportar o desacoplamento temporal. A abordagem inicial visa atender ao requisito do desacoplamento espacial utilizando identificadores livres de semântica de localidade. Logo, as associações de segurança utilizariam esses identificadores em vez de endereços IP. A fim de aumentar a segurança desse modelo, utilizamos identificadores criptográficos [7] para identificar as entidades e objetos na Internet, proporcionando a identificação de dados de forma persistente, independentemente dos protocolos utilizados, locais de armazenamento ou mecanismo de encaminhamento e roteamento. Esses identificadores criptográficos são gerados a partir da aplicação de uma função de *hash* criptográfico sobre uma chave pública, possibilitando a identificação segura e permanente de entidades na Internet.

A fim de prover o desacoplamento total das camadas de transporte e de rede, propomos uma nova camada entre as camadas de rede e de transporte chamada de *camada de identificação* [16]. A camada de identificação fornece identificadores de criptografia para a camada de transporte permitindo a identificação permanente de entidades, o desacoplamento dos conceitos de identificação e localização, e também possibilitando cenários de redes heterogêneas sem a interrupção da conectividade. A camada

⁵A autenticação explícita significa que a verificação dos parâmetros de segurança, por exemplo a assinatura digital, é feita diretamente com o provedor do conteúdo, e não com o agente intermediário que está armazenando o dado

de identificação permite o estabelecimento de uma relação de confiança explícita entre as entidades que são independentes da localização na rede, permitindo a mobilidade em redes heterogêneas [17]. Como resultado desse trabalho de investigação, concluímos que a camada de identificação possibilita a independência na camada de rede para cenários de mobilidade, resultando na separação espacial.

Com a evolução deste trabalho, notamos que o conceito de identificação poderia ser generalizado para qualquer objeto na Internet, por exemplo, uma pessoa, um objeto, um bloco de dados ou um serviço. Por exemplo, se uma entidade cria um vídeo e o envia para o YouTube, a entidade tem um identificador, o vídeo tem um outro e o site do YouTube possui um terceiro identificador. Mas do ponto de vista de segurança, há uma confiança implícita entre o provedor de conteúdo (vídeo) e o consumidor, que é o estabelecido com o site do YouTube. Neste cenário, a limitação está no fato de que a segurança não deve ser síncrona, pois a produção e o consumo do conteúdo não estão acoplados no tempo. Como uma segunda contribuição deste trabalho, coletamos os requisitos e esboçamos uma proposta inicial para autenticação de dados em redes orientadas à informação [18]. Embora o modelo anterior possibilite o desacoplamento espacial, permitindo o estabelecimento da relação de confiança fim-a-fim entre as entidades, ele é limitado para atender ao requisito de desacoplamento temporal devido à camada de identificação exigir a interação síncrona entre as entidades para a manutenção das sessões seguras. Portanto, a fim de manter a confiança fim-a-fim explícita entre as entidades, propomos um *plano de segurança* que fornece um ponto confiável de ligação entre os provedores de conteúdo e consumidores em ambientes temporais desacoplados. O plano de segurança contém estruturas de metadados que armazenam informações sobre blocos de conteúdo, por exemplo, identificadores de blocos, *hashes* criptográficos de autenticação e a assinatura digital, permitindo a autenticação direta com o provedor original do conteúdo, independentemente do servidor onde o conteúdo foi obtido⁶.

A migração da confiança da conexão para o conteúdo exige uma nova abordagem, onde a segurança não deve ser dependente de qualquer parâmetro externo, exceto o próprio conteúdo. Portanto, o conteúdo necessita ter parâmetros de segurança embutidos que são independentes do local de armazenamento, conexão ou protocolo de transporte. Para alcançar isso, nós utilizamos a Merkle Tree [19] como o modelo de autenticação inicial, pois esse mecanismo atende os requisitos de desacoplamento temporal e espacial. Propomos duas novas técnicas, a *skewed hash tree* (SHT) e a *composite hash tree* (CHT) [20] para prover autenticação dos dados de forma amortizada e atendendo aos requisitos das redes desacopladas em tempo e espaço.

A SHT estende as *Merkle trees* regulares para suportar a autenticação de arquivos de tamanho aleatório e também possibilitar a verificação dos dados em trânsito. A vantagem dessa abordagem é permitir a autenticação dos dados por dispositivos intermediários, uma vez que esse modo não é possível com protocolos tradicionais, pois os parâmetros de segurança são trocados entre os nós finais e os dispositivos intermediários não possuem nenhum mecanismo para a verificação dos dados. Uma limitação da SHT é a sobrecarga na autenticação associada à árvore de *hash* que aumenta com o tamanho da árvore. O segundo esquema, a CHT, aborda a limitação da sobrecarga no SHT, e permite a autenticação eficiente do conteúdo ao custo de uma hierarquia de verificação. A CHT possui dois parâmetros, α e h , que possibilitam a configuração e a otimização da sobrecarga na verificação assim como a configuração da hierarquia de autenticação, podendo ser aplicados em vários cenários de

⁶Neste caso, assumimos que um cliente pode obter uma cópia de diversas fontes, por exemplo, a partir de um cache, um servidor de uma rede CDN ou até mesmo um nó *peer-to-peer*. No entanto, a autenticação é feita diretamente com o fornecedor original.

redes orientadas à informação.

Nós validamos essas duas técnicas baseadas na árvore de *hash* em diferentes cenários de redes orientadas à informação, começando com a resolução de nomes no metadado [21, 22] por meio do plano de segurança, seguido do mecanismo de detecção de poluição em redes P2P [20] e autenticação paralela sobre o HTTP [23]. Finalmente, apresentamos o mecanismo de cacheamento seguro dos dados em trânsito utilizando *skewed hash trees* a fim de prevenir o cacheamento de conteúdos falsos [24, 25].

O procedimento de obtenção de conteúdo em redes orientadas à informação inicia com a resolução do nome no metadado correspondente. No entanto, o sistema de nomes baseado em DNS possui várias vulnerabilidades de segurança, incluindo ataques de envenenamento de *cache*, negação de serviço, entre outros. Uma das causas desses problemas é a confiança implícita dos clientes no sistema de nomes, onde os clientes confiam que os servidores de nomes irão resolver adequadamente o mapeamento do nome para endereços IP corretos. A tese apresenta um novo modelo de resolução de nomes onde a confiança é estabelecida diretamente entre o cliente e os provedores de conteúdo sem qualquer passo intermediário através do plano de segurança. O plano de segurança permite a resolução de nomes nos metadados, permitindo aos clientes utilizar as meta informações para recuperar os blocos de dados. Nesse cenário, usamos SHT e CHT para proverem a transferência da confiança explícita de um nome de conteúdo para a estrutura de metadados e também para cada bloco de dados.

Após a etapa de resolução do nome para o metadado, nós analisamos a aplicação dos mecanismos de árvore de *hash* em dois cenários de recuperação de conteúdo autenticado: redes P2P e autenticação paralela sobre HTTP. No primeiro cenário (redes P2P), propomos o emprego da CHT para detectar ataques de poluição parcial [26] durante a recuperação de dados da rede. Este tipo de ataque impede que os usuários recuperem corretamente o conteúdo da rede. Em poucas palavras, o ataque de poluição parcial consiste em um atacante modificar alguns dos blocos de dados para inibir a recuperação de conteúdo devido a falhas no cálculo do *checksum* original. Nesse caso, podemos utilizar a CHT para detectar blocos poluídos através do cálculo dos *checksums* criptográficos dos blocos de dados assim que esses blocos são recebidos. Através desse mecanismo, as aplicações podem detectar um bloco poluído e substituir apenas esse bloco ao invés de baixar o arquivo completo novamente. Como resultado desse trabalho, apresentamos um mecanismo para detectar poluição com o objetivo de reduzir o consumo de banda comparativamente aos repetidos *downloads* feitos na recuperação do conteúdo na abordagem tradicional.

No segundo cenário de recuperação de conteúdo (autenticação paralela sobre HTTP), utilizamos a CHT para fornecer autenticação de conteúdos provenientes de várias fontes, geralmente servidores espelhos do provedor original. Neste cenário, os clientes implicitamente confiam nesses servidores para obter os dados e, nestes casos, os clientes são capazes de apenas verificar os dados obtidos *após* a sua recuperação. O principal problema é a granularidade de verificação, ou seja, os clientes são capazes de verificar os dados logo depois de se obter o arquivo completo de todos os sítios, que podem resultar em casos em que um único servidor falso pode levar os clientes a descartarem todas os blocos de dados recolhidos a partir de diferentes sítios. Com o objetivo de fornecer um mecanismo de autenticação mais eficientes, propomos um mecanismo de verificação paralela usando a CHT [23] que permite a autenticação de dados no momento que são obtidos e solicitar somente aqueles que estão corrompidos.

Além dos cenários anteriores, a tese também investiga a possibilidade de utilizar a técnica de árvores de *hash* na autenticação de dados na camada de rede com o objetivo de viabilizar o *cachea-*

mento seguro em dispositivos denominados de roteadores de conteúdo (*content routers*). Essa proposta é baseada no fato das árvores de *hash* suportarem mecanismos de autenticação de dados sem qualquer semântica de localidade e, além disso, de não serem fim-a-fim, ou seja, qualquer dispositivo intermediário na rede é capaz de verificar os dados em trânsito desde que ele tenha as chaves corretas. Portanto, uma proposta consiste na utilização de roteadores de conteúdo [24] para verificação dos dados em trânsito [25] para prevenção de tráfego não autorizado na rede.

Organização do Texto

Além deste capítulo introdutório, a sequência do texto encontra-se organizada da seguinte forma: o Capítulo 2 discute os trabalhos relacionados aos mecanismos de autenticação, aos protocolos de segurança e às arquiteturas de nova geração. O capítulo inicia com a descrição do mecanismo de assinatura de *Lamport* e as *Merkle trees*, cujo modelo é a base para esta tese. Em seguida, descrevemos os protocolos de segurança atuais, IPSEC e TLS, destacando suas principais características e limitações para uso em redes orientadas à informação. Finalmente, apresentamos as arquiteturas de nova geração relacionadas à redes orientadas à informação.

O Capítulo 3 propõe um novo modelo de segurança baseado em identificadores seguros e uma nova camada denominada de *camada de identificação*. Os identificadores seguros permitem a identificação permanente dos nós finais separando os conceitos identificação e localização e possibilitando o suporte nativo à mobilidade. Além disso, nós estendemos a camada de identificação para suportar um protocolo de comunicação seguro chamado de IDSEC para a autenticação dos nós finais e avaliamos o modelo proposto em diversos cenários de mobilidade.

O Capítulo 4 propõe um mecanismo de autenticação para redes orientadas à informação baseado em um *plano de segurança*. Esse capítulo inicia com o levantamento dos principais requisitos de uma rede orientada à informação, seguido de uma proposta de um mecanismo de autenticação baseado em dois tipos de árvores de *hash*: *skewed hash tree* e *composite hash tree*. Além disso, propomos uma estrutura de metadados como um mecanismo seguro de armazenamento de informações de segurança no plano de segurança, e também como um ponto de ligação confiável entre os provedores de conteúdo e consumidores.

O Capítulo 5 descreve os cenários de aplicação do plano de segurança e os mecanismos de autenticação baseados em árvores de *hash*. Esse capítulo inicia com o mecanismo de resolução de nomes em estruturas de metadados, seguido por dois cenários de aplicação das técnicas de autenticação baseadas em árvores de *hash*: detecção de poluição em redes P2P e autenticação paralela de dados sobre o HTTP. Por fim, iniciamos a discussão sobre outros cenários de aplicação, tais como autenticação dos dados por dispositivos intermediários e também *cacheamento* seguro.

Finalmente, o Capítulo 6 apresenta as conclusões do trabalho, delineando as principais contribuições e resultados obtidos, como também, abordando algumas questões em aberto para serem exploradas como trabalhos futuros.

Introduction

The original Internet architecture was designed to interconnect geographically distant computers to share resources and services, for example, remote login and file transfer. The architecture was constructed over the packet switching technology to improve the robustness of the communication network, for example, to be robust against natural disasters or even a war. The basic communication model uses the Internet Protocol (IP) to exchange packets between two end-hosts, where each host is assigned a unique and permanent IP address. Whenever an end-host wants to send a packet, it creates a packet with the destination IP address and routers along the path route to the destination. Conversely, the destination could reach back the source just by inverting the source and destination IP addresses in the header, according to the *end-to-end principle* [1].

The advent of the World Wide Web (WWW) in the early 90' has made the Internet popular outside the research community, making documents and services, such as HTML pages and electronic commerce, easily accessible for users through Web-browsers and Web-servers. Meanwhile, new security requirements appeared such as application authentication and data confidentiality, especially with the emergence of online banking. Initial efforts to provide security mechanisms for applications started with the *secure network programming* (SNP) [2] library and the *secure sockets layer* (SSL) [3] that came with the Netscape Navigator Web-browser. SSL would evolve to a standard called Transport Layer Security (TLS) [4], which is currently in version 1.2 by the time of this thesis. Also, Internet Security (IPSEC) [4] was proposed to provide host-to-host security, a mode that SSL could not provide. IPSEC provides security at the network layer, such as authentication, integrity and confidentiality of the data exchanged between two end-hosts. TLS also provides security mechanisms likewise IPSEC, but it operates at the transport layer.

The introduction of new usage scenarios, such as wireless networks and information-centric networking, exposed some limitations of the current Internet architecture. One of these limitations is known as *IP semantic overloading problem* [5, 6, 7], which consists of the simultaneous usage of the IP address in both network and transport layers. At the network layer, the IP address is used as a topological identifier, indicating a geographical point on the Internet topology. At the transport layer, the IP address is used as a host identifier, being used as end-host identifier during a connection establishment between two end-hosts⁷. Hence, there is an implicit coupling between the concepts of identification and location, where an end-host's identification is bound to a network location, hindering native mobility scenarios without modifications on the end-host's identifier. From the security standpoint, the absence of a permanent and secure identifier during a host mobility event opens vulnerabilities for security attacks, such as impersonation and man-in-the-middle attacks [8].

⁷In order to establish a connection using Berkeley sockets API, an application needs a *transport level identifier* (TLI), which is composed of the source and destination IP and port and the protocol version.

A second limitation resulting from the evolution of the Internet refers to the conflict created between the current use of the Internet and its original design [9]. Currently, the Internet is predominantly used to obtain content such as music and videos, independently of their storage location in the network. For example, users access the Internet for relevant information, such as weather forecasts, news, etc., but they are not interested in the IP address from which the content is being fetched from. In fact, users are more interested in the *provenance*⁸ of the content rather than the location where it was fetched from. These networks are known as *information-centric networking* [10] or, as Van Jacobson has coined, *content-centric networking*, and are divided into two types: *clean slate information-centric architectures* and *legacy information-centric applications*. The former group represents new architectural approaches to support fully information-centric features, such as *publish/subscribe* [10] and *get/put* [11] primitives rather than regular *send/receive*. The latter group represents applications that are already deployed in the current Internet architecture and are *conceptually* information-centric, but it is implemented over legacy *send/receive* paradigm. Some examples in this group include content delivery networks (CDN) [12] and Peer-to-peer (P2P) [13]. The main focus of these networks is to obtain the content efficiently without prior knowledge of the source of the content. For example, the CDN networks perform host name resolution via dynamic DNS redirection, while P2P networks perform name resolution to content through queries to a distributed hash table (DHT) [14].

The main characteristics of these networks are the *spatial* and *temporal* decoupling. The spatial decoupling refers to the lack of knowledge of the server's address hosting the content. The temporal decoupling refers to the absence of an explicit session between clients and servers to obtain a piece of content. On the other hand, the original Internet architecture is connection-oriented between end-hosts, where the knowledge of a location is a prerequisite for obtaining the data in a synchronous way. From the security standpoint, the absence of a stable identifier and an explicit session between two hosts represents an architectural and technical challenge to provide content security in these new usage scenarios.

The security protocols in use today (e.g., IPSEC and TLS) are synchronous and use the IP address as end-host's identifier to establish a secure connection between two end-hosts. Nevertheless, in information-centric networks, there is no prior knowledge of the data source location, leading to trust problems of the source. For example, the data source can be a Web cache or a peer, but there is no explicit mechanism for authenticating this source hosts. In this case, security protocols can provide *content verification* in content-oriented networks, but not *validation*. Verification provides mechanisms to check whether a particular set of data was not changed during the transfer between two end-hosts, while validation means that the retrieved content is what the client requested for. For instance, if a user requests for content in a P2P network, even if it is possible to establish a connection using IPSEC or TLS, the user could check that the content was obtained from the source peer and it was not modified during the transfer. However, there is no mechanism to validate that the obtained content is actually the content that the user originally requested for.

This thesis proposes a new authentication mechanism using hash trees in order to provide efficient data authentication and explicitly with the original provider, and also independently of the host where the data were obtained. We propose two techniques for data authentication based on hash trees, called

⁸The origin of the data regarding of the producer, for example, whether a given piece of news comes from a trusted source.

skewed hash tree and *composite hash tree*, for data authentication in information-oriented networks. Once created, part of the authentication data is stored in a *security plane* and another part remains attached to the data itself, allowing for the verification based on content and not on the source host. In addition, this thesis presents the formal model, specification and implementation of two hash tree techniques for data authentication in information-centric networks through a security plane. Finally, this thesis details the instantiation of the security plane model in two scenarios of data authentication: 1) Peer-to-Peer networks and 2) parallel data authentication over HTTP.

Contribution of this Thesis

The main contribution of this thesis is to propose a security model for data authentication in information-centric networks satisfying the spatial and temporal decoupling requirements. We start with a security model to provide spatial decoupling in the security protocols, and then, we generalize the proposed model to support temporal decoupling as well. The initial approach aims to solve the spatial decoupling requirement by using identifiers that do not contain any location semantics. Therefore, security associations would be bound to these identifiers rather than IP addresses. In order to add security in this model, we use cryptographic identifiers [7] to identify entities and objects on the Internet, providing persistent data identification regardless of the protocols, storage or forwarding mechanisms. These cryptographic identifiers result from the hash over a public key and allow for secure and permanent identification of entities in the Internet.

In order to provide complete transport decoupling from the network layer, we propose a new layer between the network and transport layers called *identification layer* [16]. The identification layer provides cryptographic identifiers for the transport layer allowing for the permanent identification of entities, while decoupling the entity identification from the network layer, and also enabling heterogeneous networks without disrupting the connectivity. The identification layer allows for the establishment of an explicit trust relationship between entities that are independent of location on the network, enabling mobility in heterogeneous networks [17]. As result of this investigative work, we found that the identification layer enables the independence on the network layer for end-host mobility, resulting in the spatial decoupling.

As evolution of this work, we realized that the concept of identification could be generalized to any object on the Internet, e.g., a person, an object, a piece of content or a service. For example, if an entity creates a video and posts it on YouTube, the entity has an identifier, the video has another one and the YouTube site has a third identifier. But from security standpoint, there is an implicit trust between the content provider (video) and the consumer that is established with the YouTube site. In this scenario, the limitation lies in the fact that security should not be synchronous, since the production and consumption are not coupled in time. As another contribution of this work, we collected the requirements and outlined an initial proposal for data authentication in content-oriented networks [18]. Although the previous model met the spatial decoupling and provided directly end-to-end trust between entities, it is limited to satisfy the temporal decoupling requirements, mainly because the identification layer requires synchronous interaction between entities for the secure channels maintenance. Therefore, in order to keep the explicit end-to-end trust, we propose a *security plane* that provides a trusted binding point between content providers and consumers in temporal decoupled environments. The security plane contains metadata structures that hold important information about a

give piece of content, for example, block identifiers, authentication hashes and digital signature, and allows for direct authentication with the content provider, regardless of the server where the content was obtained⁹.

The trust migration from the connection to the content itself requires a new approach, where security must not be dependent on any external parameter, except the content itself. Therefore, the content needs to have its own built-in security that is independent of connection, local storage (server) or transport protocol. In order to achieve this, we use the Merkle Tree [19] as the initial authentication model, since this mechanism meets the spatial and temporal decoupling requirements. We propose two new techniques, the Skewed Hash Tree (SHT) and the Composite Hash Tree (CHT) [20], to provide amortized data authentication in spatial and temporal decoupled networks.

SHT extends regular Merkle Trees to support random size files authentication and allows for in-transit data authentication. The benefit of this approach is to enable data authentication by intermediate hosts, which is not possible with traditional protocols because security parameters are exchanged between end-hosts and intermediate devices do not have any mechanism to verify the data. One limitation of the SHT is the authentication overhead associated to hash tree as the tree size grows. The second scheme, CHT, addresses the overhead limitation in SHT, and allows for efficient content authentication at the cost of some hierarchical verification scheme. CHT has two parameters, α and h , that allows for the overhead and authentication hierarchy configuration, being able to be used in many information-centric verification scenarios.

We validated these two hash tree techniques in different information-centric networking scenarios, starting with name to metadata resolution [21, 22] in the security plane, followed by pollution detection in P2P networks [20] and parallel authentication over HTTP [23]. Finally, we apply the skewed hash tree technique for secure content caching in the network[24, 25].

The content retrieval procedure in information-centric networks start with the name to metadata resolution. However, the current naming system based on DNS has several security vulnerabilities, including cache poisoning attacks, denial of service, among others. One of the causes of these problems is the implicit trust placed by customers in the naming system, where customers trust the name servers that they will properly resolve the name mapping into IP addresses. We contribute with a new name resolution model where the trust is directly established between client and content providers without any intermediate step through the security plane. The security plane allows for name to metadata resolution, allowing clients to use the meta information to retrieve the data blocks. In this scenario, we use SHT and CHT to provide the explicit trust transfer from a content name to the metadata structure and also to each piece of data.

After the name to metadata resolution step, we go further and analyze the application of the hash tree mechanisms in two authenticated content retrieval scenarios: P2P networks and parallel authentication over HTTP. In the first scenario (P2P), we use CHT to detect partial pollution attacks [26] during data retrieval from the network, which prevents users to correctly retrieve content from the network. In a nutshell, the partial pollution attack consists of an attacker modifying some of the data chunks to inhibit the recovery of content as the checksum will not match to the original. In this case, we can use CHT to early detect polluted blocks by calculating a cryptographic checksum over the data chunks as soon as it is received. Through this mechanism, applications can detect the

⁹In this case, we assume that a client can obtain a copy from many sources, e.g., from a cache, a CDN server or even a peer node. However, the authentication is done directly with the original provider.

polluted block and replace it rather than completely re-downloading the complete file. As a result of this work, we present a mechanism for detecting pollution that can reduce bandwidth consumption due to repeated re-downloads adopted in the traditional approach.

In the second content retrieval scenario (parallel authentication over HTTP), we use the CHT to provide content authentication that comes from multiple sources, usually mirror servers of the original provider. In this scenario, clients implicitly trust these servers to fetch data and they are just able to verify the downloaded data *after* its retrieval. The main problem here is the verification granularity, i.e., clients are able to verify data just after the file is completely downloaded from *all* sites, which may lead to cases where one single fake server forces the clients to discard all pieces collected from different sites. Aiming to provide a more efficient authentication mechanism, we propose a parallel verification mechanism using CHT [23] that allows data authentication as content blocks are retrieved.

Additionally to the previous scenarios, we also investigate the possibility to use the hash tree techniques in data authentication at the network level and also in content caching. The hash tree techniques provide data authentication mechanisms without any location semantics. Additionally, the security model is not end-to-end, i.e., any intermediate device in the network is able to verify the in-transit data as long as it has the correct keys. Therefore, one possible application is to deploy *content routers* [24] to provide secure content caching, preventing from fake content storage. The hash tree mechanism can also be integrated in information-centric transport models [25], where the content retrieval logic is bound together with verification mechanism. Another application is the usage of the hash tree mechanism to authenticate data prior to its caching in the servers. Currently, Web caches do not have any mechanism to validate the stored content because there is no explicit security mechanism embedded in the data and the security parameters are exchanged between clients and servers, not with the Web-cache.

1.2 Text Organization

In addition to this introductory chapter, this thesis is organized as follows: Chapter 2 presents the background information about authentication schemes, security protocols and clean-slate architectures. The chapter starts presenting the Lamport one-time signature scheme and Merkle Trees, which is the basic authentication model for this thesis. Next, we describe the current security protocols, IPSEC and TLS, outlining their main features and limitations for information-centric networking. Finally, we present clean-slate architectures for background information about information-centric networking.

Chapter 3 proposes a new security model based on secure end-host identifiers and a new logical layer called *identification layer*. These secure identifiers allow for the permanent end-host identification, splitting the concepts of end-host identification and location, enabling native mobility support. Additionally, we extend the identification layer to support a secure handshake protocol called IDSEC for end-host authentication and evaluate the proposed model in mobility scenarios.

Chapter 4 proposes an authentication mechanism for information-centric networks based on a *security plane*. The chapter starts with the main requirements of an information-centric networking, followed by the proposal of a data authentication mechanism based on *skewed hash trees* and *composite hash trees*. Additionally, we also propose a *metadata* structure as a placeholder for security information in the security plane, and also as a trusted binding point between content providers and consumers.

Chapter 5 describes the application scenarios for the security plane and hash tree-based authentication mechanisms. This chapter starts with the name to metadata structure resolution, followed by two application scenarios of the hash tree techniques: P2P pollution detection and parallel data authentication over HTTP.

Finally, Chapter 6 presents the final conclusions of this thesis, outlining the main contributions and obtained results. Lastly, we address some open issues to be explored as future work.

Capítulo 2

Related Work

This chapter presents the background information about security protocols and the related work to this thesis. The chapter starts describing the authentication schemes, Lamport one-time signature scheme and Merkle trees. Then, we describe the traditional protocols currently used and some new protocols for data authentication. Later, we present the information-centric architectures that are related to our work and, lastly, we summarize this chapter.

2.1 Authentication Schemes

2.1.1 Lamport-Diffie One-time Signature (1979)

The Lamport One-Time Signature Scheme (OTSS) [27] is a digital signature scheme used to authenticate messages without using public key cryptography. The signature scheme relies on one-way functions, such as cryptographic hash functions, to provide efficient signature schemes with low cost. The OTSS is also capable of providing security even with the development of quantum computers because its strength is based on the difficulty of reversing a cryptographic hash function and not on the product of large prime numbers. Quantum computers are believed to factor large numbers in polynomial time, becoming a threat to current public key cryptography that relies on the product of large numbers or discrete logarithms.

The OTSS mechanism signs each bit of the message with the sender's private key, requiring that the private key has the same number of bits of the message to be signed. Any modification in the message or in the signature can be detected when the signature is checked against the sender's public key. After using the private key to sign a given message, it must be discarded and a new one must be generated to be used to sign another message.

The OTSS is divided in three phases: *key generation*, *message signature* and *signature verification*. The key generation phase involves the generation of the public and private keys. The message signature phase generates a signature with the private key over a single message. The signature verification phase authenticates a message with the sender's public key and its signature.

The private key generation uses a Pseudo-Random Number Generator (PRNG) to produce m pairs of random numbers (total of $2m$ numbers) each number with a k -bit length, as illustrated in Fig. 2.1. The private key will be the array with $2m$ random numbers organized in two columns, resulting in a total size of $2m * k$ bit long private key. The public key is generated by applying a cryptographic hash

function H over each element on the private key array, resulting in a new array with the same length and number of elements.

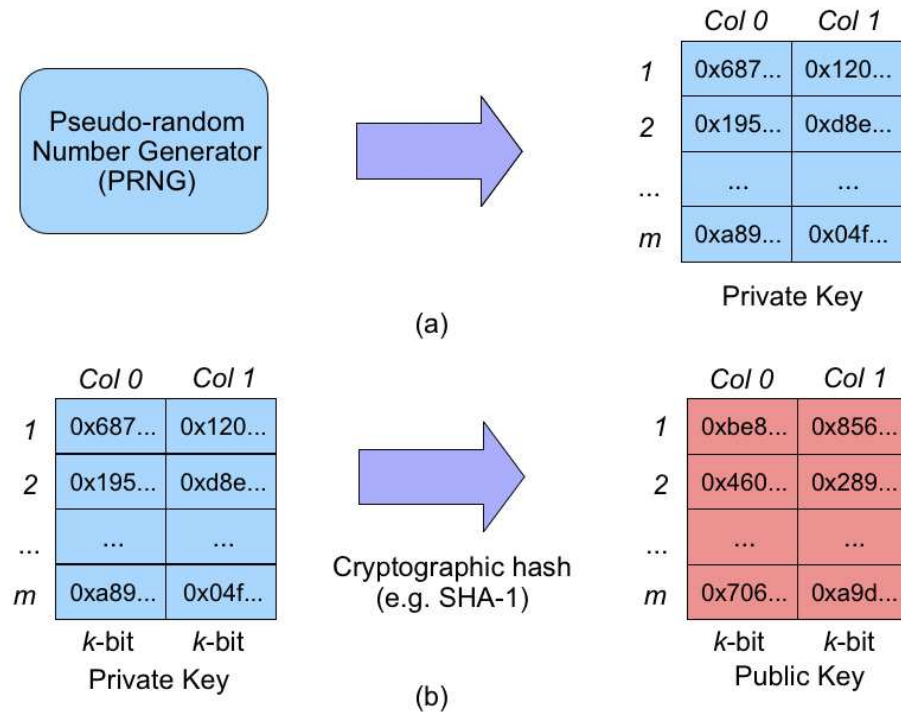


Fig. 2.1: OTSS key generation phases. (a) Private key generation using a Pseudo-random Number generator. (b) Public key generation from the private key.

The signature generation over a message using an OTSS private key starts with the message passing through a cryptographic hash function, resulting in a k -bit output. Later, for the i^{th} bit on the message digest, the OTSS signature algorithm picks up the first element of the pair of the private key if it is a 0-bit or the second element of the i^{th} pair if it is a 1-bit, as illustrated in Fig. 2.2. The resulting array of bits is the signature, which must be sent along with the message to the receiver.

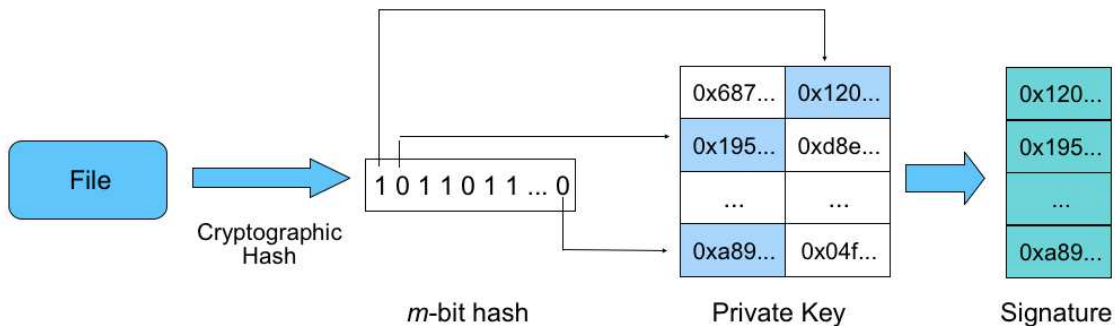


Fig. 2.2: OTSS signature generation using the private key.

Finally, the OTSS verification procedure involves the sender's public key retrieved from a public directory and the hash of the message with the same cryptographic hash function H . For each bit

in the message digest (signature), the algorithm selects the first or the second bit in the sender's public key. Finally, the receiver applies the cryptographic hash over all values in the signature and the resulting values must match the same slots in the public key. This procedure is illustrated in Fig. 2.3.

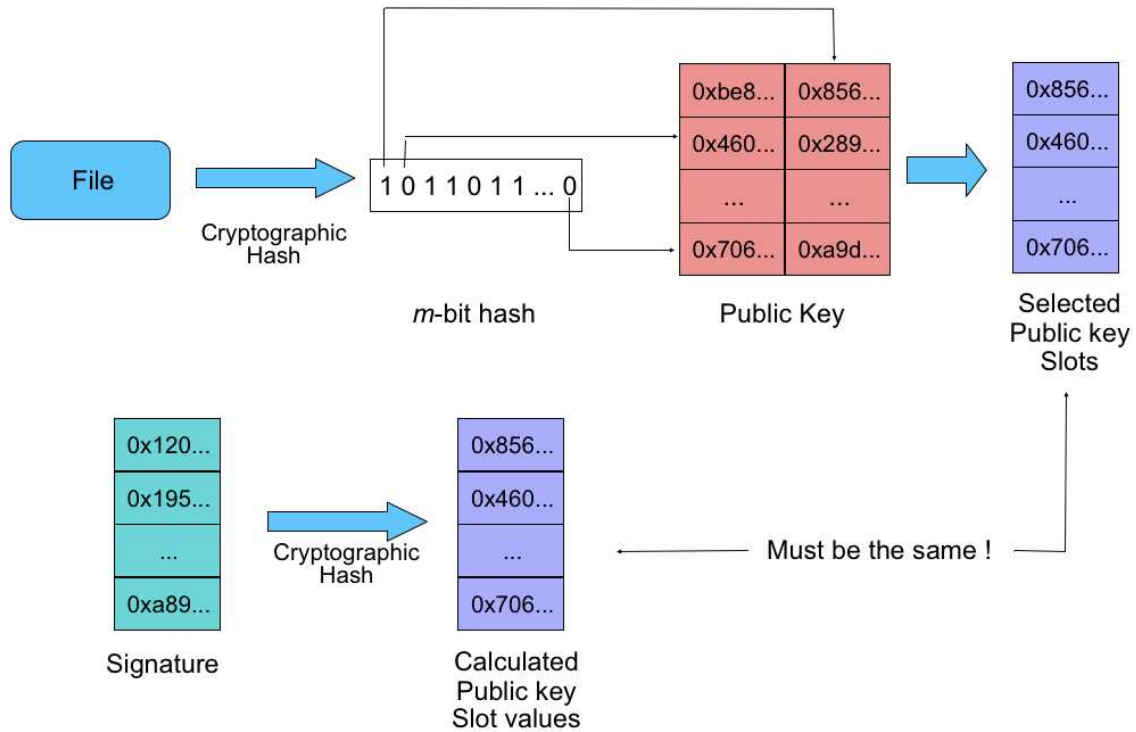


Fig. 2.3: OTSS signature verification procedure.

The drawback of the mechanism is the size of the keys. Let H a cryptographic hash function that accepts inputs of any size and generates a m -bit message digest. In this scenario the public and private keys are $2mk$ -bit long and the resulting signature is mk -bit long. As an example, considering a SHA-1 cryptographic hash function that outputs a message digest of 160 bits. Therefore, the required size for the public/private keys and the signature are 320 bits and 160 bits respectively.

2.1.2 Merkle Hash Tree (1989)

The Merkle Hash Tree [19], also known as Merkle Tree (MT), was proposed as an optimization mechanism for Lamport one-time signature scheme. As described previously, the OTSS requires one pair of public and private keys to sign a message, and the key length must be the same size as the hash of the message. The main limitation of the scheme is the one-time use of the key pair, requiring constant publication of public keys by the sender. This major bottleneck has motivated Merkle to investigate an alternative to reduce the number of keys in the signature mechanism, motivating his work on reducing the number of public keys required in the verification scheme. The main idea is to create a hash tree over a set of private keys and generate one public key over the set of private keys. The benefit of such scheme is that a sender just needs to transmit a public key for a set of private keys, reducing the number of unnecessary public key publications in the public directory.

A Merkle Tree is a complete (or balanced) binary tree where each leaf node holds the hash of a message or data block and each internal node holds the hash of the concatenation of its children's node, defined in the following equation:

$$n_{parent} = \text{hash}(\text{hash}(n_{left}) || \text{hash}(n_{right})) \quad (2.1)$$

The first step to construct a Merkle Tree is to divide a file into smaller data blocks. Then a hash function is applied over each data block and the resulted hash value is stored in each leaf. The second step is to go up towards the root of the tree by hashing the concatenation of the hash value of two children's hash values and storing the result in the parent node, until reaching the top of the tree, which hash value is also called the *Root Hash*.

The *Root Hash* is the compact representation or the signature of the set of messages or an entire file. This hash value is used to authenticate each message or data block together with the *Authentication Path*, and any modification in the data set will result in a different signature. The *Authentication Path* is the list of hash values needed by a message or data block to reach the *Root Hash*, allowing the authentication procedure. The AP value on height h (AP_h) is the value of the sibling of the node at height h from the path from the leaf towards the *Root Hash*. Fig. 2.4 shows an example of Merkle Tree with height ($H = 2$) and the AP for each data block.

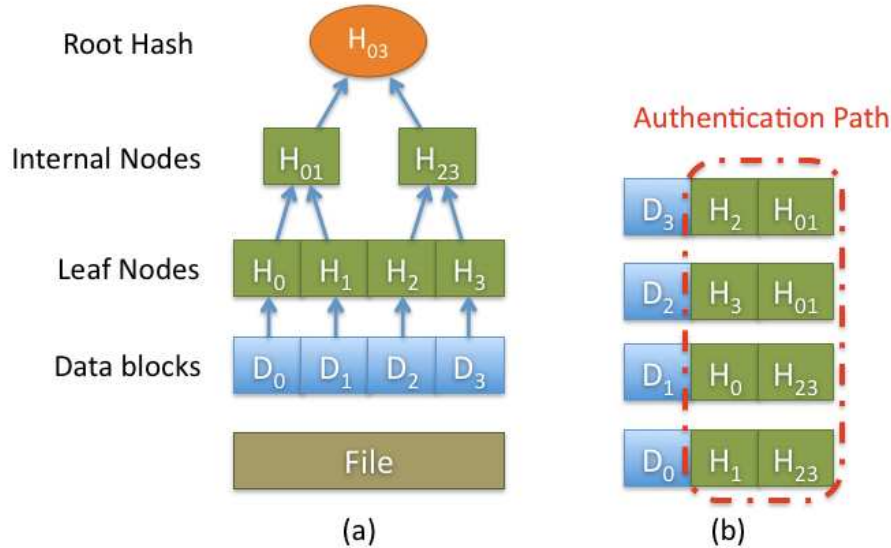


Fig. 2.4: (a) Merkle Tree of height $H = 2$ with $2^H = 4$ leaves. (b) Data blocks with their corresponding *Authentication Paths*.

The authentication procedure of data block 0 (D_0) is shown in Fig. 2.5. It is assumed that the *Root Hash* is previously transmitted to the destination prior to the authentication procedure. Then, for every data block that arrives (in this example, the first one to arrive is data block 0, even though it could be anyone), it is authenticated using the AP in the data block. The authentication procedure starts by applying a hash function over data block 0 and the resulting digest is concatenated with the first AP element, which is H_1 . Then, the hash function is applied again over the concatenated value

and the result is concatenated with H_{23} . The resulted hash value must be equal to the *Root Hash* (H_{03}) transmitted previously and, if so, the data block is authenticated.

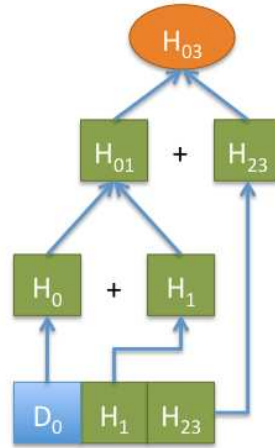


Fig. 2.5: Authentication of data block zero using Merkle Trees.

The Merkle Tree presents many appealing characteristics, such as the exclusive use of hash functions in the authentication procedure and the independent and one-to-many authentication features. As explained above, the Merkle Tree uses only hash functions to authenticate data, presenting an interesting authentication mechanism for low processing devices, since Public Key Cryptography (PKI) [28] processing is much slower than hash functions due to large prime number computations needed to perform the cryptographic functions. The Merkle Tree technique also turns possible the out of sequence or independent data block authentication, since there is no binding between the data blocks, just between the data and their *Root Hash*. This feature is interesting in file-sharing systems, where data blocks naturally come out of sequence and need immediate authentication. Lastly, the Merkle Tree offers one-to-many authentication feature, which is interesting in the context of loosely coupled systems, where one data block needs to be authenticated by many parties (e.g. file-sharing, publish/subscribe systems). Since there is not any establishment of secret between the parties (the *Root Hash* is not a secret information, actually it is publicly known), any data block could be authenticated by anyone that receives it.

Even though Merkle Trees present the previously mentioned advantages, the tree generation algorithm requires a power of two number of leaves ($2, 4, 8, \dots, 2^n$) to work. However, in the context of data authentication, the number of data blocks hardly is a power of two, since most of the files have random size. Another problem regards the length of the *Authentication Path* due the number of repeated hash values. These two problems will be addressed below.

Unbalanced Merkle Tree Problem

A Merkle Tree of height H requires a complete binary tree in order to work with the original algorithms. As a consequence, the number of leaves must be equal to $N = 2^H$. The original motivation for the Merkle Tree was to aggregate a set of public keys together to reduce the number of public key publications. In that context, the generation of private keys as a multiple of power of two was not

a strong requirement since users could generate extra keys, for example 16 in a batch, and save the unused keys for later use.

Although the number of keys generated in a batch was not an issue in the public/private key context, this technique can not be directly applied over archives. Files have random sizes and there are two simple approaches to make it fit in a regular Merkle Tree: (a) divide the file in a multiple of power of two chunks, regardless of the size of each chunk, and (b) divide the file in pre-defined chunk size and append *zero* leaves in the Merkle Tree to balance it. The first approach is not good because the chunk size can be a restriction in many scenarios, for example, network and storage discs. In the network scenario, packet sizes are usually limited by the Maximum Transmission Unit (MTU) of the network, and if the goal is to provide authentication of the network data, then the limitation is usually 1500 bytes. Storage discs also have the minimum block size that are usually multiple of 4KB. In this case, the best fit is the block size to be exactly the size of the minimum block size of the disc to optimize the block usage.

The second approach is the naive zero leaves padding of the Merkle Tree. The basic idea is to add chunks with *zeros* in the tree to make it balanced. This procedure is illustrated in Fig. 2.6.

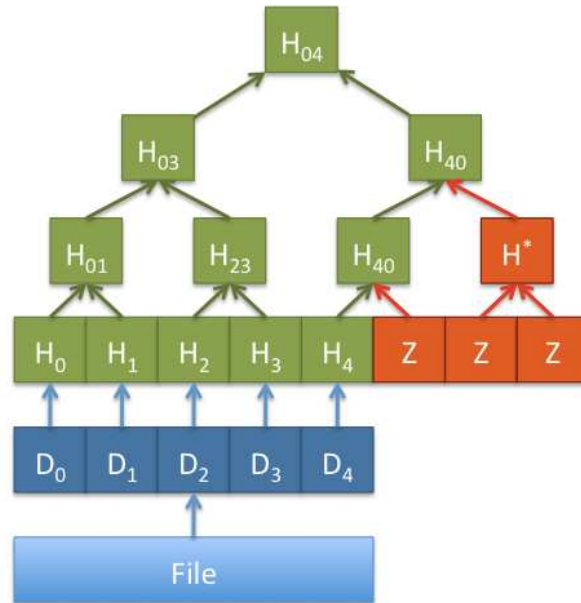


Fig. 2.6: Unbalanced Merkle Tree ($H = 4$) with naive zero padding scheme.

Although the zero padding scheme solves the unbalanced tree problem, the number of zero leaves grows exponentially depending on the chunk distribution. The worst case happens when a user has a number of blocks that fits in a balanced tree plus one, requiring a binary tree that is the double of the size. As the height of a MT grows, the number of required zero leaves increases proportionally, resulting in $2^{(H-1)} - 1$ zero leaves, when the number of blocks is equal to $N/2 + 1$, and it requires a tree with height $H + 1$ to hold the hash information of all data blocks. Hence, the number of hash function calls in the zero padding scheme is the same as in the balanced tree since the zero leaves are computed as a regular leaf. Thus, the total number of hash function calls is the sum of all hash functions calls over the N data blocks, plus in the intermediate and top nodes.

To illustrate the unbalanced Merkle Tree problem, Tab. 2.1 shows the Merkle Tree height and the range of data blocks with the corresponding file size storage for 1KB data blocks. The table shows that at height $H = 17$, the tree holds files from 64MB to 128MB, where files with size closer to 64MB will have the highest number of zero leaves and files closer to 128MB will be almost balanced trees.

Tab. 2.1: Merkle Tree height vs. number of data blocks

Tree height (H)	# of blocks	File range (MB)
16	32,769-65,536	32-64
17	65,537-131,072	64-128
18	131,073-262,144	128-256
19	262,145-524,288	256-512
20	524,289-1,048,576	512-1024

The total number of hash functions calls using the original Merkle Tree and zero padding scheme is the same as the balanced tree since the zero leaves are computed as a regular leaf. The number of hash functions calls can be calculated by summing up all the hash functions applied over N data blocks, plus in the intermediate and top nodes. Thus, we have:

$$\sum_{i=0}^H 2^i = 2^{H+1} - 1 = 2N - 1 \quad (2.2)$$

Therefore, a Merkle Tree with N leaves (where $N = 2^H$) requires $2N - 1$ hash function calls to generate the *Root Hash*, regardless of the number of empty leaves in the tree.

Merkle Tree Overhead Problem

One of the main attractiveness of a Merkle Tree is the independent authentication of each data block as it arrives in the destination. This is possible due to the *Authentication Path* (AP) embedded on each data block, which is the list of hash values needed by the data block to reach the *Root Hash*. The list of hash values needed is equal to the size of the height of the Merkle Tree, since the procedure needs the sibling hash value of the data block at each height to reach the *Root Hash*. Therefore, the number of hash values in the AP is $H = \log_2 N$, where N is the total number of data blocks and $N \log_2 N$ is the total overhead of the Merkle Tree during the file transfer.

However, a careful inspection in the Merkle Tree shows that the number of hash values available in the tree of height H is $2^{H+1} - 1$ (excluding the *Root Hash*), which are all the needed hash values to reach *Root Hash* by all data blocks. Hence, we can conclude that $N \log_2 N - (2^{H+1} - 1)$ hash values can be removed since they are repeated hash values in the AP, as shown in Fig. 2.7.

Based on Fig. 2.7, the total number of hash values in a Merkle Tree of height H can be expressed on the number of input data blocks N :

$$\sum_{i=1}^H 2^i = \frac{a(r^{H+1} - r^1)}{r - 1} \approx 2^{H+1} = 2 \times 2^H = 2N \quad (2.3)$$

Hence, the number of repeated hash values in the *Authentication Paths* is:

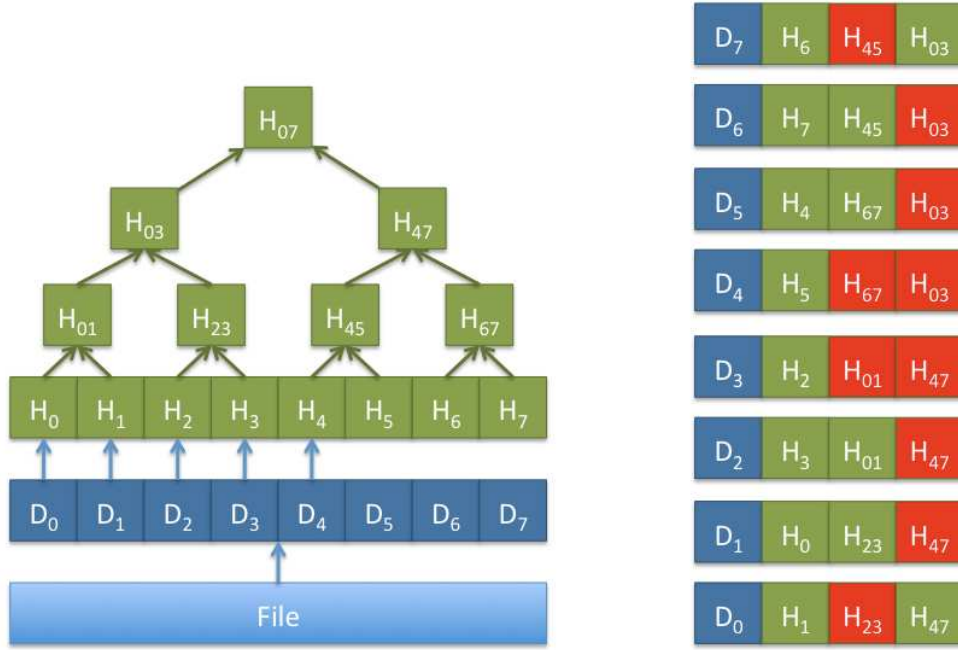


Fig. 2.7: (a) Regular Merkle Tree construction. (b) *Authentication paths* associated to each data block. The repeated hash values are highlighted in red.

$$N \log_2 N - 2N = N(\log_2 N - 2) = N \log_2\left(\frac{N}{4}\right) \quad (2.4)$$

Fig. 2.8 shows the percentage of repeated hash values versus the height of the Merkle Tree. For Merkle Trees with height $H \geq 4$, the number of repeated hash values in the *Authentication Path* is more than 50% of the total hash values.

The hash values repetition in the AP is an important characteristic of the original Merkle Tree to maintain the total independence between signed messages, since they may not be correlated. In the context of data authentication, where the idea is to authenticate data blocks that are a subset of a large file, we propose to weaken the independence requirement of each data block in the Merkle tree, since there is already an intrinsic relation between all data blocks (they are part of the entire content). The motivation for our work is to create an authentication data structure that presents similar properties to the regular Merkle Tree but with lower authentication overhead. As described in the last section, the Merkle Tree authentication overhead grows $O(N \log_2 N)$ mainly due to the repetitions in the AP values. This repetition is important to maintain the total independence between data blocks, allowing any sequence reception and authentication.

Applications

Merkle trees have been employed in many different scenarios, such as HTTP authentication, P2P networks and multicast. In [29], the authors use Merkle trees to authenticate HTTP responses. Each HTTP response from a server has an AP attached, allowing clients to verify the response. However, the proposed approach does not take into account responses that generate a skewed tree. ALPHA [30]

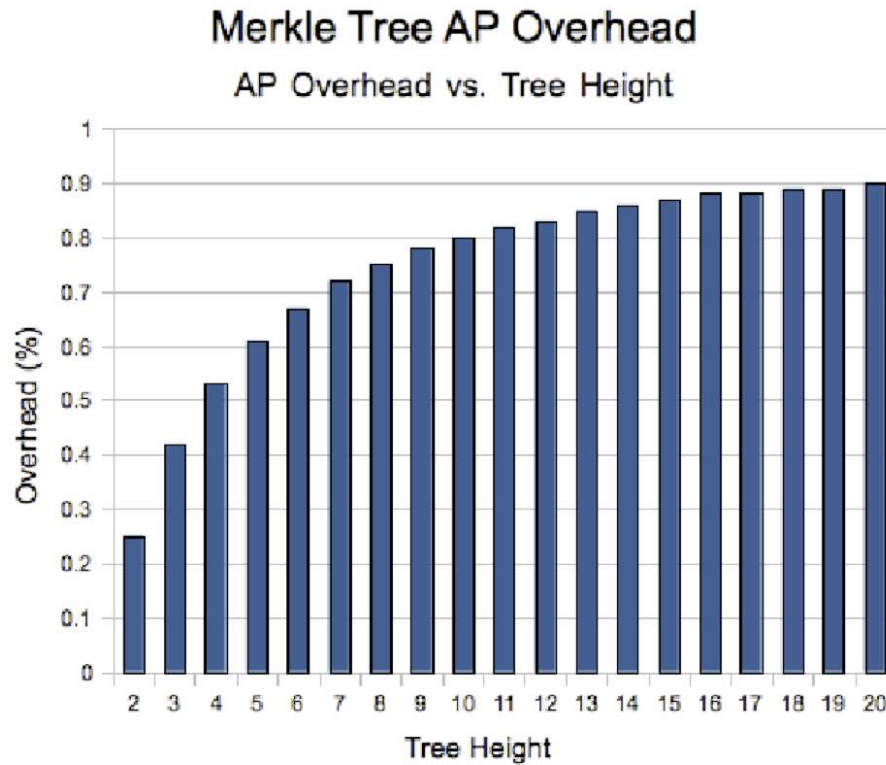


Fig. 2.8: Percentage of repeated hash values in the AP versus the height of Merkle Tree.

uses Merkle Trees to create pre-signature data, reducing the total number of required signatures per bulk of exchanged messages. However, it does not take into account the fact that Merkle Trees require balanced trees in order to work properly. In [31], the authors propose an authentication mechanism to validate streaming data retrieval. Each block has an AP, allowing for data verification and also for data recovery in case of a missing block using erasure codes. However, they always use balanced Merkle trees, which may be a constraint in case of data block fragmentation. In [32], the authors propose a mechanism to authenticate peer-to-peer data using an *authenticated DHT*. The ADHT stores the AP of all data blocks, allowing peers to retrieve and authenticate data chunks. Nevertheless, it still requires a balanced tree to create the APs, resulting in zero-padded trees.

There are also some optimizations for the Merkle Tree traversal algorithms. In [33, 34, 35], the authors optimize the authentication path generation, improving the overall time and storage parameters. In this work, we use the original algorithms proposed by Merkle due to its simplicity and elegance. However, we can still support these extensions to optimize the signature and verification procedures.

2.2 Standard Security Protocols

In this section we describe the main features of the Transport Layer Security (TLS) and Internet Protocol Security (IPSEC) protocols.

2.2.1 Secure Sockets Layer (1994)

Secure Sockets Layer¹ was proposed as a security protocol to protect the transactions in the Web and is the predecessor of the Transport Layer Security [36] (1999). The primary goal of TLS is to provide privacy and reliability between two communicating applications. The protocol is divided into two layers, where the lowest layer uses the SSL Record protocol and the upper layer uses the TLS handshake protocol. The SSL Record protocol works over TCP and provides security mechanisms to encapsulate upper layer protocols. The TLS handshake protocol allows for servers and clients to authenticate each other and to negotiate encryption algorithms and cryptographic keys before the application sends or receives any data. One benefit of TLS is that it works over the transport layer (TCP) and supports any application in the upper layer. SSL has three basic properties:

- **Authentication.** Client and server can be authenticated using public key cryptography using the digital certificates exchanged in the handshake;
- **Integrity.** Connection is reliable and tamper proof. Messages exchanged between client and server uses keyed MAC and secure hash functions to provide data integrity.
- **Confidentiality.** Connection is private, i.e., encryption is used after the initial handshake to exchange a secret key. Thus, further communication uses symmetric cryptography for data protection;

TLS was designed to be transparent to applications and works directly over TCP, using the transport level identifier (TLI)² to identify each connection between applications, as illustrated in Fig. 2.9(a). Some benefits of the TLS include the transparent NAT traversal and widespread support by the web-browsers, making it one of the most popular security protocols used nowadays.

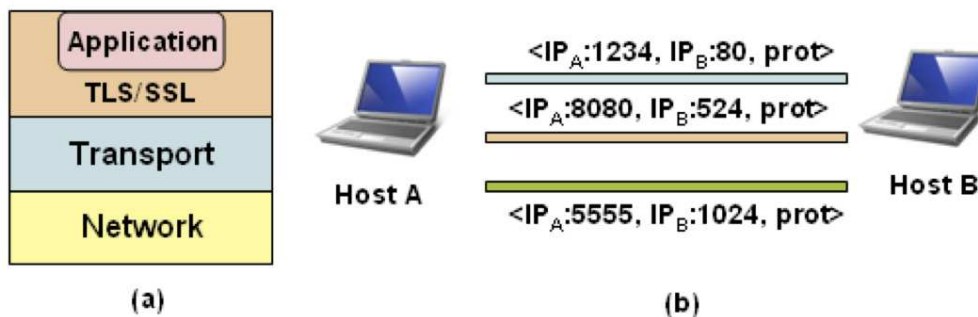


Fig. 2.9: (a) TLS between the transport and application layer. (b) Multiple secure channels between the same end-hosts.

On the other hand, some limitations of TLS include the need of multiple secure channels between the same end-hosts, transport protocol dependence and TLS support by applications. As each secure

¹Specification available online: <http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt>

²A connection from an application using sockets requires a TLI to identify an end-host. A TLI is composed of a tuple with source and destination IP addresses and ports and the protocol.

channel is indexed by a TLI, it is required a new secure channel between applications even if they are running on the same pair of end-hosts, as shown in Fig. 2.9(b). Another restriction of TLS is the dependence on the TCP. As TLS do not work with unordered packets reception, which may happen with UDP, the TLS rely on the TCP to deliver ordered segments of data. Finally, all applications using TLS must support the protocol, leading to problems with legacy applications if the version is unsupported. These limitations are actually consequence of the Internet architecture, for example, TLS relies on TCP, which was constructed around end-points. Hence, it must be attached to IP addresses and a requirement to use TCP as the transport layer.

2.2.2 Internet Protocol Security (1998)

The Internet Protocol Security (IPSEC) [37] is a suite of protocols that offers security services for the IP layer, both IPv4 and IPv6, including access control, connectionless integrity, data source authentication and limited traffic flow confidentiality. Most of the security services are provided by two security modes, the transport mode with Authentication Header (AH) and the tunnel mode with Encapsulating Security Payload (ESP), and also key management protocols. In the AH mode, IPSEC protects all data above the network layer, providing end-host authentication and message integrity, although the payload is not encrypted. From the network stack perspective, IPSEC inserts a new header between the network and transport layers, as illustrated in Fig. 2.10(a). In the ESP mode, the entire IP packet is encapsulated inside a new IP packet, provides authentication, message integrity and data confidentiality, protecting the entire payload, as illustrated in Fig. 2.10(b). The transport mode is mainly used in communications between end-hosts and the tunnel mode in Virtual Private Networks (VPNs).

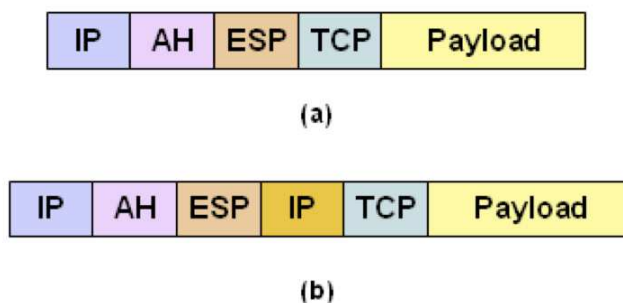


Fig. 2.10: (a) IPSEC transport mode. (b) IPSEC tunnel mode.

IPSEC operates in a host as a security gateway or as an independent device. The protection levels offered by IPSEC are based on the policies stored in the Policy Database (SDP), which is managed by a system administrator or application. There are three main policy modes that can be applied to each packet: PROTECT, DISCARD and BYPASS. PROTECT protects all packets using IPSEC; DISCARD rejects all packets from a range; and BYPASS allows the traffic to go through the IPSEC node. Each protection policy is represented by a *security association* (SA), which is the simplex connection that provides security services in one direction. To secure a typical bidirectional communication, a pair of SA is required. The SA is stored into a SA Database (SAD) and it is indexed by the IP address.

Some benefits of the IPSEC include the support of security ignorant applications and single tunnel between the same pair of end-hosts. On the other hand, it also has some drawbacks. One of them is related to the incompatibility of the AH and Network Address Translator (NAT) [38], described in [39]. The IPSEC AH uses the source and destination IP address to calculate the Keyed MAC [40] of the protected message. Packets passing through a NAT will have their source IP address changed, resulting in a different HMAC in the destination. Thus, packets are discarded and the communication is not maintained between the parties.

2.3 New Security Protocols

This section presents some proposals of security protocols, outlining their main features. We analyze the Packet Level Authentication (PLA) mechanism and the Host Identity Protocol (HIP). The former model (PLA) proposes to sign each packet with a digital signature allowing for data chunk verification in the network. The latter one (HIP) introduces a new security protocol to enable the security communication between entities rather than end-hosts. We will describe each one below.

2.3.1 Packet Level Authentication (2005)

Packet Level Authentication (PLA) [41] proposes a mechanism to authenticate data using digital signatures on each piece of data. The main idea is to provide a hop-by-hop and end-to-end authentication mechanism, where each packet can be verified on each hop and also in the end-hosts, protecting the network infrastructure and providing network availability. The technique resembles a bank note, which contains built-in security mechanisms, such as watermarks, holograms, etc, but in PLA, each packet contains the digital certificate from the issuer, timestamp and the digital signature. PLA assumes that public key cryptography can be used to sign large data flows, allowing for invalid or unwanted traffic detection in the next hop.

2.3.2 Host Identity Protocol (2006)

Host Identity Protocol (HIP) [42] introduces a new authentication and key exchange protocol with protection capabilities against denial of service (DoS) attacks. The protocol introduces a new name space located between the network and transport layers called *host identity* to solve the IP semantic overload problem. The name space is compounded of *host identities* (HIs) which are the public keys of a pair of asymmetric keys. Each end-host has one or more HIs associated, and, in order to limit the size of the HI, HIP proposes the Host Identity Tag (HIT), a 128-bit identifier resulted from the cryptographic hash of the HI. The HIT is a static identifier and dynamically binds to the network layer, providing mobility and multi-homing support.

The authentication and key exchange protocol employs a puzzle mechanism to challenge the hosts before the key exchange to avoid any denial of service attacks. In this case, the Initiator must compute some resource-consuming problem before engaging in the key exchange procedure. After solving correctly the puzzle, the hosts engage in a Diffie-Hellman key exchange procedure and establish a secure key between the end-hosts.

The HIP working group also proposed some extensions to support new features, such as data protection [43], mobility support [44] and name resolution [45]. The HIP data protection extension uses the IPSEC Encapsulating Security Payload (ESP) to protect the underlying communication channel. After HIP securely authenticates and establishes a symmetric key between the parties, the protocol forwards the security parameters to the underlying IPSEC ESP to apply the selected security policies.

2.3.3 An adaptive and lightweight protocol for hop-by-hop authentication (2008)

An adaptive and lightweight protocol for hop-by-hop authentication (ALPHA) [30] combines a set of security mechanisms to create an authentication protocol for wireless and sensor networks. Wireless networks are susceptible to flooding and interception attacks and they are usually restricted regarding CPU, memory and energy consumption. They combine hash chains techniques to provide strong binding between sources and destinations after hash anchors have been securely exchanged. In addition, ALPHA protects data traffic using HMAC, where the key is the corresponding hash chain element. Finally, ALPHA uses Guy Fawkes protocol [46] to provide an interactive delayed secret disclosure for integrity protection and authentication for unicast streams. Guy Fawkes is an interlocking scheme where a sender publishes an encrypted message, e.g., some event that may happen, and after the event has happened, the sender discloses the secret key to claim the ownership of the event.

The ALPHA protocol works as follows: a sender and a destination securely exchange a pair of hash chain anchors to be used one on each direction. Then, for each message that is going to be sent, the sender gets the next element in the hash chain and use it as the HMAC key. But prior sending the message to the next hop, the sender sends the MAC of the message and waits for the ACK from the intermediate hop. Once the sender receives it, it sends the message and discloses the hash chain element used in the HMAC. ALPHA uses Merkle Trees to reduce the verification overhead on each hop by reducing the number of hash chains required on each hop. The main limitation regarding ALPHA is the memory requirement since it needs to buffer messages in the interlocking protocol. Another limitation is the bandwidth because each message passing through a hop needs three other messages prior sending to the next hop.

2.4 Clean Slate Internet Architectures

2.4.1 A Data-oriented (and Beyond) Network Architecture (2007)

The Data-oriented (and Beyond) Network Architecture (DONA) [11] proposes a new clean-slate architecture for the Internet aiming at the name persistence and content availability. The authors state that the majority of the Internet usage is for data retrieval and service access, whereas the Internet architecture was originally designed around host-to-host applications. Despite the fact that current architecture can support data access, it has some limitations from the host-to-host model to support all features. Some problems include the name persistence, content availability and data authenticity. Current DNS names are built around hierarchical names that represent administrative domains, for example, *mail.acme.org*, and whenever a piece of data is transferred from one server to another, its name is also changed as consequence of the mobility event. Therefore, names are not persistent through transfer between network locations. The second issue is the content availability, where the

goal is to retrieve pieces of content from the closest available source. Finally, the third issue regards the content authenticity. Since content can be retrieved from any source (e.g., network caches, peers and servers), it is important to have security mechanisms to authenticate these data.

DONA proposes to replace the DNS names with a flat, self-certifying names to enable the name persistence and authenticity, and the DNS resolution mechanism with a name-based anycast primitive to handle content availability. The names work around *principals* (P) and *labels* (L) and have the P:L format. Principals are the data owners and they are the only ones that can offer to serve and provide access to the data. Labels are the tags given by the data owner to the content and owners are responsible for the label's uniqueness. Therefore, each datum comes with a metadata including the principal's public key and principal's signature over the datum, and it is represented by a *triplet* data, public key and signature.

DONA's name resolution mechanism aims at high availability by finding close-by copies and also avoiding faulty nodes. There are two basic primitives: FIND(P:L) and REGISTER (P:L), where find retrieves a copy of the data using an anycast primitive, and register registers a data using the given P:L. On each domain, there is a *resolution handlers* (RH) that maintains a registration table mapping a name to both next-hop RH and the distance to the copy. The RH are configured similarly to the current hierarchical topology in the Internet and FIND/REGISTER messages follow the same path towards the core of the Internet. Any RH on the path having the requested object, or willing to store it, answers with the requested data or saves it.

2.4.2 Publish-Subscribe Internet Routing Paradigm (2008)

The Publish-Subscribe Internet Routing Paradigm (PSIRP) [10]³ is a clean-slate approach to apply pure publish/subscribe [15] paradigm in the entire communication system, from the simple link layer node discovery to the application data delivery. Each data element in PSIRP is called a *publication* and it has two identifiers associated: a scope Id (SId) and a rendezvous Id (RId). The SId defines the access rules to the publication, similar to what virtual private networks (VPNs) represents today but in the application level. The RId is a static and unique publication identifier, allowing its global identification across the Internet. Both of them are free of location semantics, thus, they can co-exist in many places in the network without conflict of location. Whenever a user wants to receive a publication, and its update as well, it sends a SUBSCRIBE(SId, RId) to the network and publishers can publish data using the PUBLISH(SId, RId, data) primitive to the network. The network works as a blackboard and is able to store data within it. As a consequence, it is possible to reduce data transfer by delivering closer copies of the same data to the subscribers.

There are three main roles associated in the publication and subscription matching and delivery: *forwarding layer*, *topology manager* and *rendezvous server*. The forwarding layer uses a Bloom-filter based source routing mechanism to deliver data from a publisher to any subscribers. The topology manager is responsible for computing the bloom-filter containing the route from a publisher to a subscriber taking into account the closest possible source. In this scenario, only the topology manager has the complete view of the topology, thus, it is able to compute the path between publishers and subscribers. The rendezvous server is the entity responsible for receiving PUBLISH and SUBSCRIBE messages and do the matching between them. The server stores a set of tables with the pending subs-

³The current continuation of this project is known as PURSUIT. Available online: <http://www.fp7-pursuit.eu>

criptions and the incoming publications. Whenever there is a match, it generates an event that triggers the topology manager to generate a bloom-filter with the path between publisher and subscribers. Finally, this bloom-filter is sent to the publisher which publishes the publication in the network, and the network is responsible for the publication delivery to all subscribers.

From the security perspective, PSIRP uses the packet level authentication model as the default security mechanism to provide data authentication. Prior to a packet delivery or reception, each publication chunk is digitally signed to provide content authentication and protect against unauthorized modifications in the network.

2.4.3 4WARD Project (2008)

The 4WARD [47] project proposes a clean slate architecture based on the concept of Network of information (NetInf). NetInf is a communication architecture for networking of information, helping users to manage information differently by classifying logical groups and making it available anywhere in the Internet. Content in the NetInf are divided into three groups: bit-level objects (BO), data objects (DO) and information-objects (IO). Bit-level objects are the representation of the raw data in bits, i.e., a file, a data chunk. A data object is a set of bit-level objects. An information object is a set of data objects combined with a metadata that describes the data objects.

Whenever a user wants to access an object, he performs a semantic search in the NetInf dictionary to get a list of object descriptions that matches the user's criteria. Each element in the list is associated with a data object ID and users request for a data object ID to get a list of bit-level objects. The access to information objects are controlled by access rights, which define the type of access, e.g., read/write and execute, and who can access the data. In order to provide the object access, the metadata includes a list of public keys of the entities that can access the resource. The access to the information objects in the network is enforced by the *security controller*, who is in the border to the NetInf network. Prior to the access to the information, users perform a public/private key challenge to verify whether a given user has its public key in the metadata structure of the requested data. As an extension to support application names [48], NetInf also uses a naming scheme based on DONA's name, with a principal and a label. In this case, they changed the label to point to the information object that contains the metadata of the requested content. The metadata may not contain the owner's public key, but the public key of a proxy to preserve *anonymity* of the content publisher.

2.4.4 Content-centric Networking (2009)

Content Centric Network (CCNx) [49, 50] and the current continuation of the project, Named Data Network (NDN)⁴, proposes a clean slate architecture for content-oriented communication driven by consumers' *interests*. The founding principle is that users should connect to pieces of content (look for content) rather than to hosts. In addition, many of the popular content are produced once and copied many times to the different clients. Requests for pieces of content are identified by hierarchical names, similar to fully qualified domain names, where each name is composed of a high-level application name, concatenated together with the version, segmentation index and the cryptographic hash of the content to provide data integrity. These requests are routed directly on the names towards the server.

⁴Available online: <http://named-data.net>

Network caches on the path are able to identify and respond with the cached data on behalf of the server. NDN relies on application-level identifiers to identify pieces of content in the network, content resolution and forwarding is bound to the hierarchical structure described in the URL.

From the security perspective, NDN uses packet level authentication to provide content authentication, integrity and provenance using digital signature using public key cryptography and Merkle trees to amortize the verification overhead. In addition, NDN uses content centric encapsulation to provide request anonymity and privacy. The concentric encapsulation mechanism is the recursive application of encapsulation technique using public key cryptography. Each NDN router advertises its public key, so content providers can use it and use that key to encrypt the last part of the name. For example, given a NDN URL */ndn/uc/alice/music/*, the content provider can use the first NDN router's public key to encrypt *music*, resulting in */ndn/uc/alice/ $E_{alice}(music)$* . This procedure can be applied recursively using the public key at each hop on the path. As the data traverses the path, it is decrypted and the signatures are switched to the correct ones. Also, publishers can use a symmetric key prior encrypting the name, so they separate the symmetric encryption from the encapsulation mechanism. Another option is to use the full encryption mechanism, hiding the entire URL. For example, given the URL above, a publisher could encrypt it to generate $X = E_{alice}(/ndn/uc/alice/music, k)$. The resulting encrypted URL could be re-encapsulated again to $Y = E_{Bob}(/ndn/cs/bob/X, k')$. Finally, the URL could be published as */ndn/usp/anonymous/Y*.

2.5 Summary

The Chapter 2 described the related work for this thesis. We have presented the original proposals for the Lamport One-time signature scheme and how it motivated the work on Merkle Hash Trees. Then, we described the basis for the Merkle Tree and its limitations to support random file size authentication and the growing overhead problem. Later, we described the current security protocols, IPSEC and TLS, and explained their limitations to support mobility and content authentication. In the last part, we described some clean slate information-centric architectures that support new content retrieval procedures.

Capítulo 3

Towards Identity-centric Security

This chapter presents an identification-based security model to overcome the dependence on network attachment locations of the current security protocols. We start with the motivation of our work, outlining the main issues and consequences on the dependence of network location information. Then, we present the *identification layer* approach as an architectural alternative to detach the end-host identification from its underlying network location. Finally, we extend the identification layer to support security mechanisms (e.g., authentication, data integrity and confidentiality) adapted to mobility scenarios.

3.1 Motivation

Two commonly used security protocols today are the Internet Protocol Security (IPSEC) [37] and Transport Layer Security (TLS) [36], providing secure mechanisms to authenticate and protect sensitive data exchange between hosts over the Internet. IPSEC operates on the network layer by extending the IP packet header and employs IP addresses as end-host identifiers, protecting all upper layers and also the network layer. TLS security protocol works on the application layer and establishes a secure channel between end-hosts applications based on the end-host's Transport Layer Identifier (TLI)¹ to identify the secure channel end-points. These two protocols attend most of the current demand for secure communication over the Internet: Virtual Private Networks (VPNs) and electronic commerce (e-commerce).

Despite the fact that these security protocols work properly in the commonly used scenarios, they inherit an architectural limitation from the IP protocol called *IP semantic overload* problem [6] [51] [42]. The IP semantic overload problem refers to the use of the IP address with two semantics, *identification* and *location*, despite the fact that there is just one single field to store these information (the IP address field). The IP address is used on both network and transport layers. In the network layer, the IP address is used as topological identifier, representing a host's location in the Internet topology. In the transport layer, the IP address is also used as host identifier for all incoming and outgoing connections². For each new TLS connection, the security protocol creates a new TLI to be

¹The TLI is composed of a tuple with the source and destination IP addresses, source and destination ports and the protocol number.

²Both TCP and UDP use the source and destination IP addresses from the IP header to create a pseudo header used in

used as secure end-point identifier.

These two concepts, identification and location, may get into conflict because an end-host's identification (who the end-host is) is attached to its current location in the network (where the end-host is). If we bring this situation to the real world, and making an analogy between end-hosts and people, then a similar situation would be naming each person based on her current location, for example, the street where she is currently located. As a consequence, it is hard to permanently identify a person if she moves to another location. In security scenarios, this issue is more critical because it is hard to authenticate a person that moves frequently and certify if the corresponding person is the same one in a previous location.

End-host identification using IP addresses were acceptable in the early ages of the computer machines, when the devices were large pieces of hardware that was almost impossible to move from one place to another. However, due to the introduction of portable devices, such as PDAs and smartphones, the IP address became an ephemeral characteristic of a host as mobile devices could connect to different network attachment points. Therefore, the original trust placed in the IP address is not valid because end-hosts can no longer be identified by its previous IP address. In addition, as a consequence of the IP semantic overload problem, ongoing connections could not be maintained because of the network-level mobility. End-hosts need to change their IP address as they move to new network attachment points (new IP address in the visited network), resulting in changes in the TLI that breaks all connections³.

The IP semantic overload problem has consequences in the end-host trust establishment. Originally, the trust between end-hosts was established between the network attachment points represented by their IP addresses. However, with the rise of mobility and private networks, the trust establishment is not complete anymore, since we use an ephemeral end-host information to establish the relationship, resulting in security vulnerabilities for both end-hosts. Another direct consequence of the semantic overload issue is the lack of native mobility support. As we bind the end-host location with its identification, it is not possible to change an end-host location due to a mobility event (change in the underlying IP address) and keep the same end-host identifier (same IP address).

In order to tackle the IP semantic overload problem and also the lack of stable end-host identifiers, we propose a new namespace to fill this gap called *identification layer*, described in the next section.

3.2 Identification Layer

As discussed previously, the absence of permanent identifiers for end-hosts introduces several issues, for example, secure trust establishment and mobility. In order to deal with this limitation, we propose a new namespace composed of permanent end-host identifiers called *identification layer* [16]. This namespace is composed of semantic-free identifiers and their main role is to provide permanent and unique end-hosts identifiers over the Internet. The identification layer is placed between the network and transport layers to fill the semantic gap of identification present in the current network stack. The proposed layer aims to tackle the following issues:

- **Lack of permanent identifiers.** There is no permanent end-host identifier for applications,

the checksum computation, albeit in modern network stacks, the checksum is not verified.

³Actually, the operating system flushes all ongoing connections with the old IP address from the connection table.

leading the transport level to *borrow* the network-level topological identifier (IP address).

- **Secure end-host identification.** Adding security semantics to the permanent end-host identifiers.

In order to satisfy the first requirement, we propose the use of random length identifiers that may be generated by any end-hosts. However, this mechanism alone is limited for providing security because anyone could generate any random identifier and it would be hard to authenticate that. Thus, in order to attend the second requirement, we propose the use of cryptographic identifiers [7] to identify end-hosts due to its intrinsic security properties. Cryptographic identifiers are generated using a strong cryptographic hash over an end-hosts public key, binding end-hosts identification with an internal property of the end-host. Additionally, these identifiers allow for *self-certification*, which is the process to certify if a given public key is bound to a certain end-hosts. This procedure is important to transfer the trust from a public key certified by a trusted certificate to the network-level end-host identifier, allowing end-hosts to authenticate other end-hosts identifiers.

The identification layer is located between the network and transport layers, thus, providing secure identifiers for the transport layer while detaching the network level. Fig. 3.1(a) illustrates the standard TCP/IP stack, where the socket bindings use the IP address in both network and transport layers. Fig. 3.1(b) illustrates the identification layer inserted between the network and transport layers, providing stable identifiers for the upper layers while it dynamically binds to the network layer.

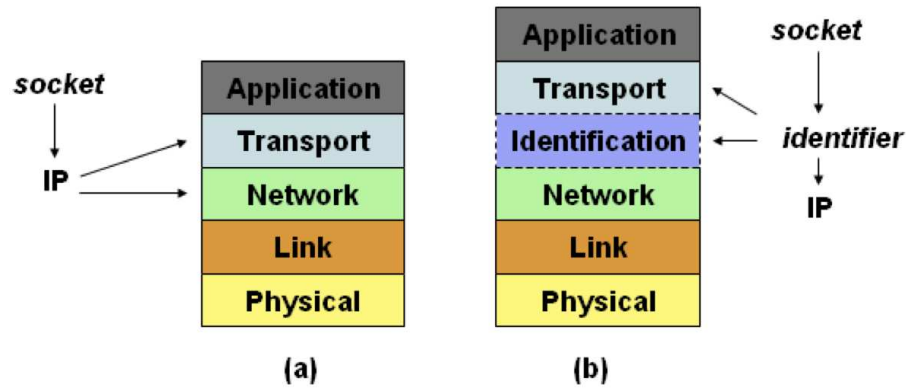


Fig. 3.1: (a) Standard TCP/IP protocol stack. (b) Protocol stack with the identification layer.

The main argument to use identification layer is to migrate the trust from the network-level identifier (IP address) to the end-host itself, represented by the cryptographic hash of its public key. As a consequence, end-hosts can be identified independently of its current location, resulting in i) secure end-hosts identification and ii) native mobility support [17]. The security of cryptographic identifiers lies on the security of cryptographic hash functions and their resistance against pre-image attacks⁴. Therefore, security level of cryptographic identifiers can be increased based on the cryptographic hash

⁴Pre-image attacks are attacks against the hash functions that aims at finding collisions in the hash namespace. They are separated in two types: first pre-image and second pre-image attacks. The first pre-image attack consists of an attacker that knows a hash digest h of a message M and trying to find its corresponding message that generated that hash digest. The second pre-image attack consists of finding different messages that result in the same hash digest, for example, given a message M_1 and its hash digest h_1 , try to find a message M_2 such that $hash(M_2) = h_1$.

function that is used to generate these identifiers. Note here that only the cryptographic identifiers are not enough to provide trust between end-hosts. It is necessary external mechanisms for the initial trust establishment between identifiers, for example, a previous exchange of digital certificates that are certified by a trusted third party. We are going to explore this issue in the next sections.

Another important issue regarding the identification layer is backwards compatibility. The identification layer provides secure identifiers for the transport layer, therefore, the length of these identifiers must be compatible with the parameters expected for the socket creation. We propose the use of 128-bit long identifiers resulting from the SHA-1 cryptographic hash function that is truncate to 128-bit long⁵ to be compatible with IPv6 addresses.

In the next sections, we extend the identification layer to support other security mechanisms, such as data authentication, integrity and confidentiality, and also secure mobility support.

3.3 Identification Layer Security Model

In this section, we propose a security model over the identification layer called *identification layer security* (IDSEC) [16]. IDSEC proposes a set of secure routines to authenticate and protect the communication channel through an authentication protocol with denial of service resistance capabilities and security techniques to protect the integrity and confidentiality of the communication. The IDSEC protocol works within the identification layer, providing security for all layers above the network layer. Thus, the network layer is detached from the transport layer, becoming just a forwarding agent. Some benefits of this approach include support to heterogeneous networking scenarios, such as IPv4 and IPv6 or bridging between IP world with flat routing schemes. The native data protection in the identification layer allows the complete independence of the network layer, allowing complete communication protection even in heterogeneous networks. The adoption of a security model embedded in the identification layer presents some advantages, such as:

- **Support of security ignorant applications.** Legacy applications do not need any modification in their source code to implement security mechanisms;
- **Security associations based on the end-host identifiers rather than ephemeral identifiers.** Legacy applications bind to secure and permanent identifiers rather than transient IP addresses (e.g. during mobility event or private network addresses), resulting in a better mobility support by the identification layer;
- **Heterogeneous network support.** As the security mechanisms are embedded within the identification layer, the communication between end-hosts located in different network technologies, e.g., different forwarding technologies of clean-slate architectures, is transparently supported.

3.3.1 Security Model Proposal

The IDSEC security model is composed of an authentication and key exchange protocol over the identification layer. The authentication protocol is based on the Diffie-Hellman key exchange proce-

⁵Although the hash length is the same, the truncated SHA-1 digest with 128 bits is stronger than the pure 128-bit resulted from the MD5 hash function.

ture and IDSEC extends it to add some protection capabilities against denial-of-service attacks. The end-host authentication procedure is based on the public key cryptography, where each end-host generates a pair of asymmetric keys and signs the messages with their private keys to be checked against their public keys. Thus, the true owner of the private key will be able to correctly sign the messages, allowing the correct authentication of the peer's identity. However, this approach is not completely safe against man-in-the-middle attacks, where an attacker stays between the communication channel of two real end-hosts, accessing and forwarding all data between them. One solution to prevent this kind of attack is to use digital certificates issued by a *trusted* certificate authority (CA) to authenticate the end-hosts identities. Another one is to use the concept of *leap-of-faith* [52], where users use a *weak* authentication mechanism to establish an initial trust in the other peer.

IDSEC is also extendable to support data integrity and confidentiality through the deployment of these new services in the identification layer. The advantage of such approach is that all security parameters are based on the end-hosts cryptographic identifiers and not in the IP addresses, natively enabling new scenarios such as mobility and heterogeneous networks. Based on the user preferences, security mechanisms such as message authentication codes and symmetric key cryptography can be integrated in the key exchange protocol, protecting against unauthorized access or modification of the exchanged information. The trust model in IDSEC is explicit, i.e., it is directly established between entities holding a public key. This is the main difference from the current trust model based on the IP address. Whenever there is a change in the network attachment point (e.g., due to a mobility event), the trust originally established between two cryptographic identifiers remain, regardless of the location of the current end-hosts in the network. On the other hand, within the IP-based end-host identification, there must be other external auxiliary mechanisms to restore the trust between these end-hosts.

Fig. 3.2 presents the IDSEC 3-way handshake used to authenticate and exchange security parameters between two entities. The handshake follows the principles described in [7, 53] to prevent against security threats, including resource exhaustion attacks, eavesdropping and impersonation. We differ slightly from their original proposal to reduce the original number of messages from four to three to speed up the handshake protocol while satisfying the security requirements⁶. In the following steps, *Alice* and *Bob* will be the entities in the key establishment protocol and *Eve* the malicious attacker. Note here that we do not address end-hosts, but entities, meaning that entities can be at any end-hosts.

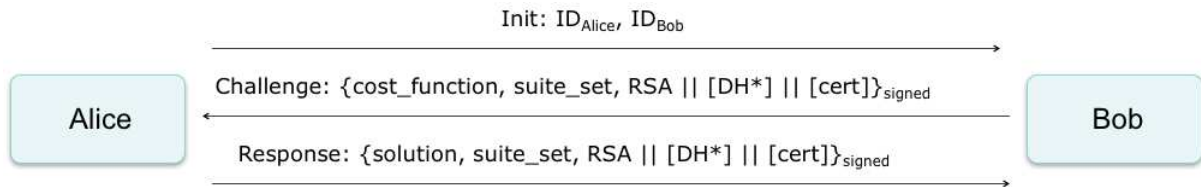


Fig. 3.2: Security association establishment protocol.

The authentication protocol uses three messages to establish a secure channel between the end-hosts: *init*, *challenge* and *response*. Alice sends the *init* message to Bob containing her ID and Bob's

⁶The original HIP base exchange is composed of four messages, where the last message is a ACK from the responder acknowledging the reception of the message and indicating that he has accepted the proposed cipher suit. Compared to our approach, we do not have this acknowledge message, but after the base exchange, we directly start the connection. If there is a loss in the third message, our protocol restarts the handshake after a time out period.

ID, where ID represents the cryptographic hash over their public keys. When Bob receives the *init* message, he stores Alice's identifier for a period of time and sends the *challenge* message back to Alice with a CPU-processing cost function. This function is used to check the commitment of the peer entity to the key establishment protocol and prevent resource exhaustion attacks (e.g. denial-of-service attacks). For a genuine entity engaged in the protocol it is reasonable to spend some CPU processing before going through the handshake, but for a malicious attacker, it is quite hard to launch a denial of service attack due to the asymmetric characteristic of the cost function. Fig. 3.3 shows an example of cost function.

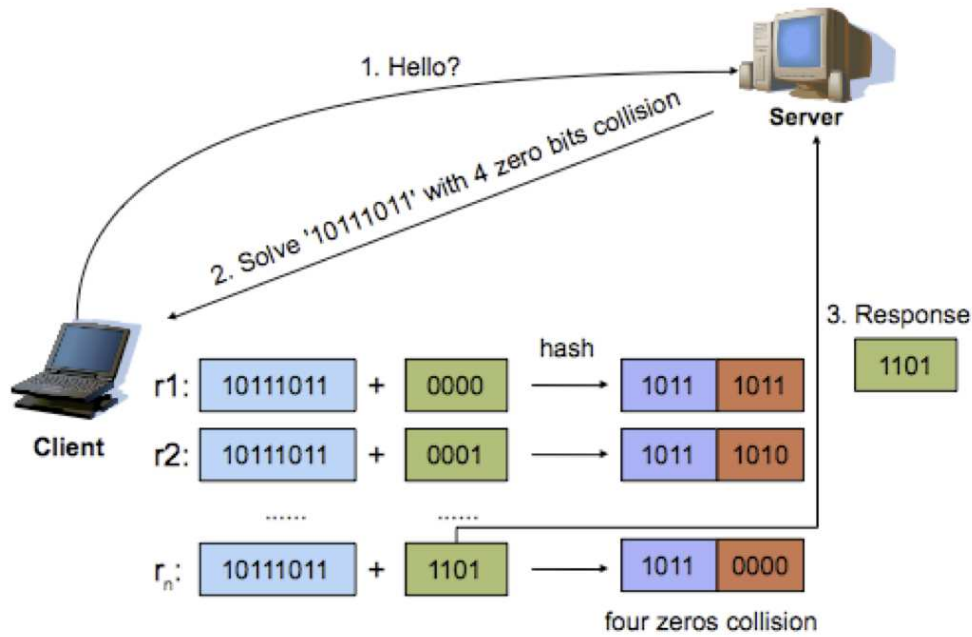


Fig. 3.3: Cost function example used to increase the costs of a denial-of-service attack.

The IDSEC proposal adopts the cost function concept proposed in Hashcash [54], where the cost function consists of brute force computations involving cryptographic hashes to find a target number of zeros to prevent resource exhaustion attacks, for example, TCP SYN attack [55]. This security threat involves a malicious attacker sending a volume of connections that cannot be completed, exhausting the internal cache of uncompleted connections. In the previous scenario, Bob sends a *challenge* message that Alice must solve through brute force computations as a proof-of-work and also that Alice is committed to the connection. To solve the function, Alice must hash a substring until it reaches a k number of zero bits, defined by Bob, which becomes the solution of the function. As there is no solution other than the brute force tentative, Alice is forced to spend some CPU processing to find some string matching the server's conditions. In the case of an attacker (Eve), she will need to spend much more CPU processing to solve the function than the targeted victim (Bob), who can check with a single hash function. On the other hand, Bob can easily check the solution with just one hash function. Thus, denial of service attacks launched against Bob can be repelled, even if many attackers are involved as in the case of distributed denial of service attack. Note that the proposed protocol can repel direct attacks against the end-host, but does not prevent the network flooding with

undesirable traffic resulted from the denial of service attacks. The *challenge* message also contains the security parameters, e.g. cipher suit proposal, used to establish the secure channel between the end-hosts, Bob's public key or his digital certificate and, optionally, the Diffie-Hellman parameters to exchange security parameters with perfect forward secrecy properties. Bob signs the message with his private key and sends the message back to Alice.

The hosts use the RSA public key algorithm to provide end-host authentication through a challenge-response mechanism to prove the ownership of the correspondent private key. This procedure involves a host challenging a peer host to decrypt a message encrypted with the peer's public key. If the peer host is able to decrypt the message and send the correct content back to the challenger, then it proved that it has the ownership of the correspondent private key. During the challenge-response procedure, the host may send the keying material encrypted with the destination public key.

One option to exchange keying material is through the Diffie-Hellman key exchange protocol to provide the Perfect Forward Secrecy (PFS) characteristics to the communication. The PFS property ensures that a session key derived from a key exchange protocol is not compromised, even when the previous key used to establish the session key is compromised. In this scenario, where the Diffie-Hellman algorithm is used to exchange the keying material for the secure channel, the previous exchanged messages can not be decrypted, even if one or both private keys are compromised. This measure is efficient against passive attackers that eavesdrop all messages exchanged and try to discover the private key to decrypt all messages. Once one of the private keys is compromised, they can derive the other keys used to protect the messages.

The Diffie-Hellman parameters generation involves the computation of large prime numbers, which may require some amount of time and CPU processing. In order to avoid any processing cost on the destination side, we propose the adoption of the static Diffie-Hellman algorithm. The static approach differs from the ephemeral Diffie-Hellman because the later one generates the Diffie-Hellman parameters every time it starts the protocol, while the former one generates the parameters just once and uses the same for all key exchanges. Although the host does not compromise any processing before checking the challenge, the host will later need to compute the symmetric key, which will result in some processing cost. One option to avoid such resource spending is the use of public key cryptography to exchange the keying material. Even though it does not provide PFS, it has lower processing costs and also may attend some specific scenarios such as the transactional ones with timestamps, like the SSL protocol today.

After Alice solves the challenge, she sends a *response* message containing the solution of the cost function, the raw keying material encrypted with Bob's public key, the security parameters, the Diffie-Hellman parameters (if requested by Bob) and Alice's digital certificate. After Bob verifies the correctness of the function, he computes the keying material and uses it to derive the keys needed to provide the other security functionality deployed over the identification layer. All the following traffic is protected with the security policies established by the handshake protocol.

The first benefit of embedding security on the identification layer is that all signaling is restricted to, or above, the identification layer. As a consequence, the maintenance of a session is not dependent on the forwarding network technology, but on the stable identifiers provided by the identification layer. Therefore, the network layer is responsible for the packet forwarding and delivery and does not need to identify the end-hosts. This property is interesting for heterogeneous networks, since network domains with different technologies can be only bridged with network-level translator and without interfering in the upper layers. This scenario is not possible in IP domain, since applications cannot

bind to different versions of the protocol to communicate in the same network. The introduction of the identification layer provides a homogeneous naming framework, allowing the natural connectivity between hosts over heterogeneous networks since the identifiers are technology independent.

The formal security analysis of the IDSEC handshake is left as future work. The primary goal of the IDSEC proposal is to validate the location decoupling model using the identification layer and that security mechanisms can use the identification layer as the forwarding fabric for the security parameters.

3.3.2 Protocol Implementation

In order to evaluate the IDSEC proposal, we instantiate the security model over the framework proposed in [56, 57]. The framework is composed of a set of components that provide specific services, for example, flat routing and identification header handling, and it is illustrated in Fig. 3.4.

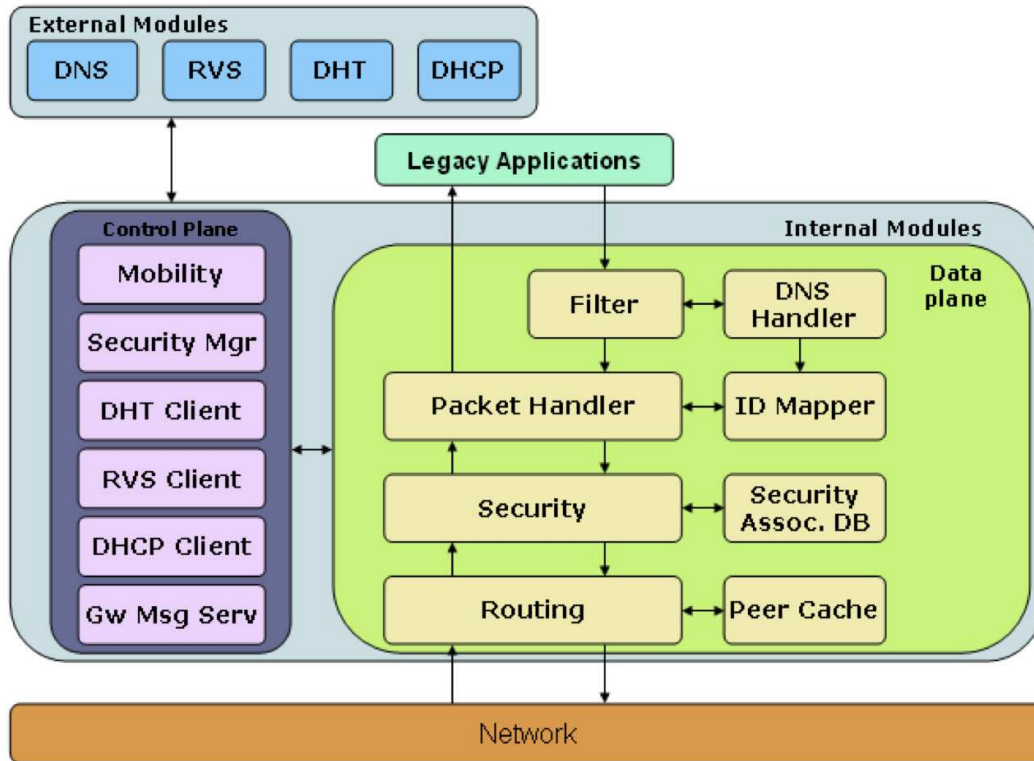


Fig. 3.4: New Internet architecture framework.

The framework features are divided into external and internal modules. The external modules are components of the architecture that provide services for the end-hosts. The internal modules are components that provide services within each end-host divided in data and control planes. Data plane modules are involved in the data delivery, identification management and flat routing. The control plane is responsible for signaling messages exchanged between components of the architecture, such as registry and redirect messages.

The external modules provide services for the nodes of the architecture, such as legacy name resolution, identifier to locator resolution and service discovery. The legacy name resolution is separated in two steps: the resolution of a name (Fully Qualified Domain Names - FQDNs) into identifiers and the resolution of an identifier into one or more locators. The service discovery mechanism is provided by a DHCP server in each domain, which must be modified to return additional parameters, such as the default router and the local RVS.

In order to fulfill the security requirements of IDSEC, three extra modules were added: *Security*, *Security Association Database* and the *Security Manager*. The *Security* module is responsible for the security policy enforcement, e.g. symmetric key cryptography for each packet sent or received by an end-host. The *Security Association Database* is responsible for the storage of all policies negotiated by the *Security Manager*, such as the algorithms used for each end-host, public key and expiration. Finally, the *Security Manager* is responsible for the three way secure handshake previously described, storing the security parameters in the *Security Association Database* to be used by the *Security* component.

In order to protect the communications between end-hosts and the external modules such as Domain Name Service (DNS), we propose secure methods to add or modify entries in the DNS and in the location manager named Rendezvous Server (RVS). The RVS is a database which holds information about mobile node's current locator in order to keep the node reachable after a mobility event. Secure insertion or modification in the DNS are provided by the adoption of the DNSSEC [58] extension. As DNS holds fairly static information about identity, nodes must authenticate themselves before any modification in the DNS server, preventing attacks such as DNS cache poisoning. In the RVS, all mobile nodes must establish security associations before any insertion or modification. This association is negotiated during the bootstrap process or whenever a node arrives at a new domain. Nodes send their public key or the digital certificates to peer nodes to verify the authenticity, and later Diffie-Hellman parameters can be used to establish a symmetric session key, providing end-to-end authentication and confidentiality.

Figure 3.5 shows the Identification Layer Security Header used in the framework. The header has a 4-bit field for the version, 4-bit field for the message type, 8-bit reserved field for further use (e.g. quality of service), 16-bit field for the payload length, 32-bit field for the security index, 32-bit field for the sequence number, a variable size field for the authentication data (e.g. HMAC), four 128-bit fields to store the source and destination pair of end-host identifiers and end-host gateway identifiers. Finally, the legacy packet is carried in the data field. The security index field is used by the end-hosts to identify the security policies used by the party and the sequence number prevents replay attacks.

The end-to-end communication is enabled by the use of source/destination identifiers, creating a secure channel between the end-hosts. This approach is useful when we consider heterogeneous networks, where the network does not deliver packets from the source end-host to the destination end-host, but from the source end-host to the boundary of the network domain.

3.3.3 Experimental Evaluation

We evaluate IDSEC from two perspectives: protocol efficiency and functionality. From the protocol efficiency, we evaluate the overhead due to the introduction of a new layer in the network stack, and from the functionality perspective, we evaluate whether IDSEC supports regular communication and mobility scenarios.

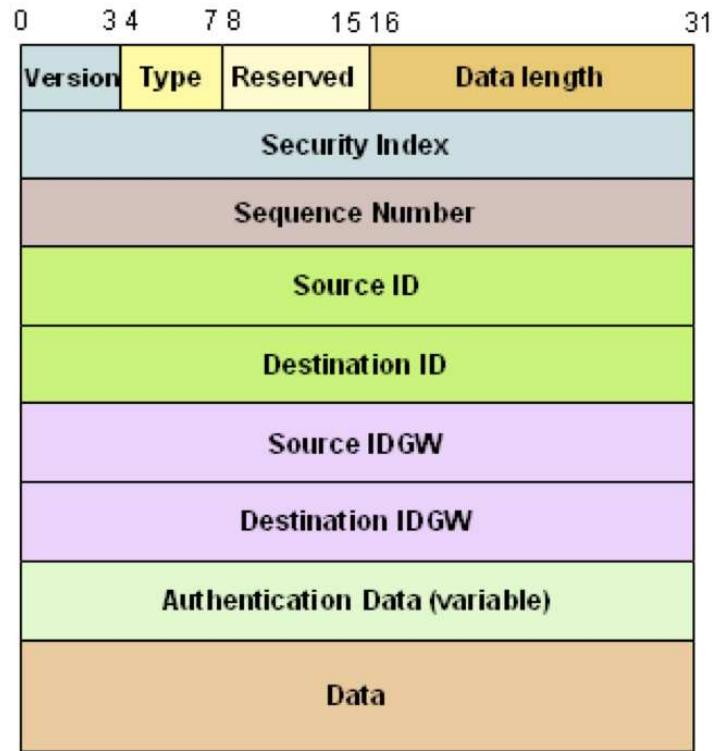


Fig. 3.5: Identification Layer Security Header.

For the protocol efficiency evaluation, we used three cipher suits to validate the IDSEC performance. The selected cipher suits are described below⁷:

- IDSEC_RSA_WITH_NULL_SHA1 - This mode provides authentication through RSA public key algorithm and message integrity with HMAC-MD5, but does not provide confidentiality.
- IDSEC_RSA_WITH_BLOW_SHA1 - Similar to the previous one, but with confidentiality provided by the symmetric key cryptography.
- IDSEC_DHE_RSA_WITH_BLOW_SHA1 - This mode provides authentication through RSA public key algorithm and message, integrity with HMAC-MD5 and confidentiality with symmetric key cryptography using Blowfish algorithm. Note that the keying material for the message digest and symmetric cryptography is generated by the Diffie-Hellman algorithm.

For the sake of simplicity, we call the first, second and third modes as *level-1*, *level-2* and *level-3* respectively. For the evaluation scenarios, we used the following hardware: two Pentium Core 2 Duo, 1.8 GHz Gb RAM, Linux Debian and NIC 100 Mb/s. For the throughput measurement, we used the *Distributed Internet Traffic Generator* (D-ITG) [59] and collected the average values of 25 samples.

⁷Note that these cipher sets were chosen just for the prototype evaluation. Other cipher sets and security services can also be deployed over the identification layer.

The first experimental evaluation analyzed the overhead due to the identification layer together with the security protocols overhead. We compared the total network throughput without any security mechanism and all the three security levels described above. The evaluation set-up consisted of starting the prototype without any security mechanism enabled and then, we added the security levels 1, 2 and 3 to compare the overhead of each level. These security levels are selected by the authentication protocol during the cipher suite selection, but the users can also select the desired security level. Hence, the user has the control over the security levels for different communication channels based on the trust on the peer. This feature also allows the introduction of control lists, such as white lists for known identifiers, e.g., internal to an organization; and policies, such as lower security requirements for frequently accessed hosts.

For this experiment, we set the constant packet rate transmission to 9140 packets/s and varied the payload size against the IDSEC security levels to evaluate the security model behavior. The packet transmission rate of 9140 packets/s was chosen because it was the maximum transmission rate with packet size of 1350 bytes that did not have packet loss. Table 3.1 summarizes the maximum throughput for each security mode considering a 95% confidence interval. As we expected, the increase of the cryptographic functions reduces the total throughput due to the processing time and packet overhead.

Tab. 3.1: Security model overhead vs. IDSEC modes (95% Confidence Interval)

Mode	Average (Mb/s)	Min. (Mb/s)	Max. (Mb/s)
Raw IP	90.72	89.11	92.34
Prototype	86.75	86.21	87.29
IDSEC - level 1	84.60	84.27	84.93
IDSEC - level 2	84.39	84.05	84.73
IDSEC - level 3	80.02	79.20	80.84

For the functionality perspective, we evaluated the prototype in mobility scenarios using legacy applications with IDSEC. For this experiment, we set up a scenario with four Pentium Core 2 Duo, 1.8 GHz, 4 Gb RAM, Linux Debian, NIC 100 Mb/s and two access points IEEE 801.11g, shown in Fig. 3.6. One machine had a Siemens IEEE 802.11g wireless interface attached to the USB interface to act as the mobile node.

For the mobility support experiment, we configured the D-ITG traffic generator to send a UDP constant bit rate of 20 Mb/s during 60 seconds between a static sender and a mobile receiver. For each security mode, we ran the experiment 25 times and the average values and standard deviation were computed. While the mobile node is receiving the traffic (Fig. 3.6 - step 1), the mobile node moves to other subnet and attaches to the new subnet access point (Fig. 3.6 - step 2). The mobile node receives new subnet parameters from the DHCP server, such as the new IP address and the default gateway. The mobility event has basically two times associated: the layer-two association time, related to the hardware association time, and the reconfiguration time, related to the time it receives the DHCP parameters and reconfigure all network parameters. The transmission rate value was chosen according to the maximum bandwidth of the wireless network with packet loss under 0.3%, in order to avoid interference in the handover time and packet loss measurement. Table 3.2 shows the prototype's handover time involving different mechanisms, such as the layer-2 handover time, layer-3 handover time (DHCP parameters retrieval), the prototype without IDSEC and level-3 IDSEC security model.

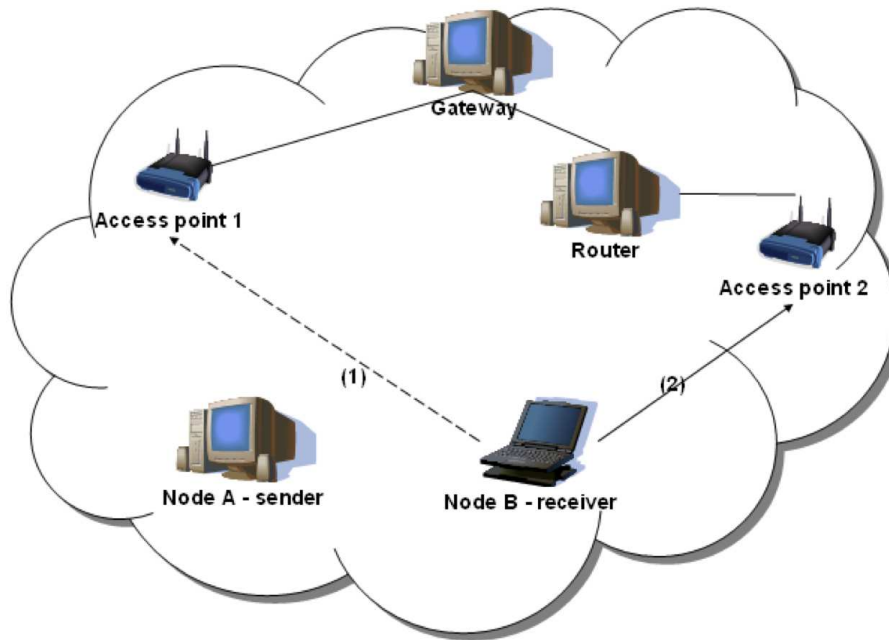


Fig. 3.6: Mobility evaluation scenarios. In (1), node B is attached to the access point 1 to communicate with node A. In (2), node B moves and uses the access point 2 to communicate with node A.

Tab. 3.2: Handover time vs. IDSEC security modes

Mode	Average (std. dev.) (milliseconds)	Pkt loss (std. dev.) (percentage)
Layer-2 handover	32.64 (2.43)	-
Layer-3 handover	35.75 (6.75)	-
Prototype	77.04 (6.70)	0.81 (0.17)
IDSEC level-3	78.62 (7.72)	0.85 (0.22)

The results show the handover latency with the security model enabled in the prototype is slightly higher than the prototype without the security model due to the secure handshake procedure and cryptographic functions. As the bandwidth in wireless networks is lower than wired networks, the impact of the security model is reduced in wireless networks compared to wired ones.

3.4 Summary

This chapter has presented the identification layer approach and how it can be an alternative to solve the IP semantic overload problem. The identification layer uses cryptographic identifiers to provide permanent and secure end-hosts identifiers in the Internet. These identifiers have intrinsic security properties that allow for end-host authentication regardless of its current location in the network. Additionally, we have presented IDSEC, the identification layer security model to support security features within the identification layer. IDSEC provides a key exchange protocol with

denial-of-service attacks capability resistance together with standard security protocols features, such as authentication, integrity and data confidentiality.

Capítulo 4

Data Authentication in Information-centric Networks

Chapter 4 presents the data authentication mechanisms for information-centric networking. First, we start with the motivation of the work, discussing the issues regarding the time and location decoupled systems. Then, we sketch the initial proposal of a security plane for data authentication in information-centric networks, outlining main requirements for a separate plane. Finally, we propose two authentication mechanisms, the *skewed hash tree* and the *composite hash tree*, that can be used in information-centric networks.

4.1 Motivation

Despite the fact that the identification layer model could solve some of the problems in the current Internet regarding security and mobility, it is limited to provide security in content-oriented network. In the current Internet, there are already many applications that look for data rather than end-hosts, for example, news feeds, videos, audios, social networks, etc, and the primary goal is to retrieve data regardless of its location (e.g., if it is in a data-center, or a cache, a peer, a host, etc). Some examples of these networks include peer-to-peer [13] networks and content delivery networks [12]. In these networks, the end-hosts are just simple data producers, consumers or holders (data storage), and the concept of data security changes slightly. One main difference is that secure connections do not guarantee that data is authentic because the trust is not transferred anymore from the end-hosts to the data. The main reason for that is because the end-host storing the data may not be the sole owner of the data anymore since it can be a simple data storage (cache) for some content provider. As a consequence, the trust on the end-host cryptographic identifier is not *explicit transferred* to the data within a connection because i) the end-host may not have produced the data and ii) there is no binding between end-host's identifier and the data itself. In this scenario, the end-host is a transient data storage server that is providing data from a content provider and may not be responsible for the data authenticity and integrity.

Therefore, we want to generalize the security model to provide data security rather than security the connection since many applications use the content-oriented paradigm. In IDSEC, the motivation to use cryptographic identifiers is to provide data security that is verifiable towards a trusted end-

host, which is generally the data generator. In content-oriented networks, end-hosts are data source or consumers, i.e., some end-hosts have applications that request for data and other end-hosts have applications that consume data. Therefore, the main focus is on the content, and this is the motivation for our work to generalize the concept of end-host identification to content identification. By identifying content, we are able to verify data towards the original producer rather than a local node that is just a server, but it also presents other challenges.

Additionally to the security issues, the increasing demand for highly scalable infrastructure for efficient content distribution has stimulated the research on new architectures and communication paradigms, where the focus is on the efficient content delivery without explicit indication of the resource location. One of this paradigms is known as information-centric networking (ICN) and its main focus is on data retrieval regardless of the source at the network level. This scenario usually happens when content providers (e.g. Warner Bros and BBC News) produce information (movies, audios, news in a Web page, etc.) and hire delivery systems such as Akamai¹ to deliver their content to the customers. The main reason to *outsource* the delivery mechanism is that content providers are not specialized in networking but on content generation, thus, it is more interesting to hire an external company to provide the delivery solution. In this model, there is a decoupling between content generation and the server storing the content itself (the actual machine serving the content to clients). Originally, servers used to generate and deliver data to the clients, however, nowadays data may be generated in specialized locations and placed in strategically placed servers in the network to speed up the content delivery to content consumers.

From the security perspective, the decoupling of data production and data hosting opens up new challenges to content authentication. One of the challenges regards the trust establishment for content authentication and a second one is the time decoupling between content consumption and production. Previously, data was generated in servers and the authentication of the server where the data was fetched resulted into an *implicit* authentication of the data because the content producer is the same as the content server. Nowadays, a common scenario is the separation between content generation and delivery, breaking the previous trust relationship established between the serving host and the content. Servers are deployed by content delivery companies to deliver data according to a contract, thus, there might not be a correlation between serving host and the data itself.

The second issue regards the time decoupling between data consumption and production, which is a direct consequence of content production and hosting separation. Content providers produce news (e.g. news feeds) that may not be synchronously consumed, i.e., BBC News web-site produces news every 5 minutes, but clients access the data after some period of time. As a consequence, content providers and consumers are *decoupled in time* and *synchronization*, and there might not be any interaction between clients and servers to ensure the content authenticity². While in the send/receive paradigm, the receiver firstly authenticates the sender before fetching the content, there may not be any knowledge about providers in the information-centric systems. As a result, some threats such as fake and unauthorized content publication or content data blocks corruption may appear. Therefore, these systems require a new security model focused on the content itself rather than securing the connection. We will discuss the information-centric paradigm and outline the main characteristics of such systems below.

¹<http://www.akamai.com>

²Sometimes the original content provider is not online to provide authentication data.

4.2 Towards Information-centric Networking

According to [15, 60], content-oriented networks have three main characteristics: *location decoupling*, *time decoupling* and *synchronization decoupling*. Location decoupling refers to the fact that providers and consumers are not directly connected; time decoupling refers to the fact that data generation is not bound to data consumption (they are not time correlated); and synchronization decoupling refers to the fact that data consumption is not synchronized with the source³. Based on these three requirements, we sketch an initial security model based on a new plane responsible for providing security features in information-centric networks. Likewise control, data and management planes, we aim to separate the security features in a separate plane to better handle the security services called *security plane*, as illustrated in Fig. 4.1. This plane acts as a secure broker that separates the authentication data from the forwarding functionality, allowing clients to retrieve pieces of content from the closest server or network caches while they fetch the authentication data from the security plane.

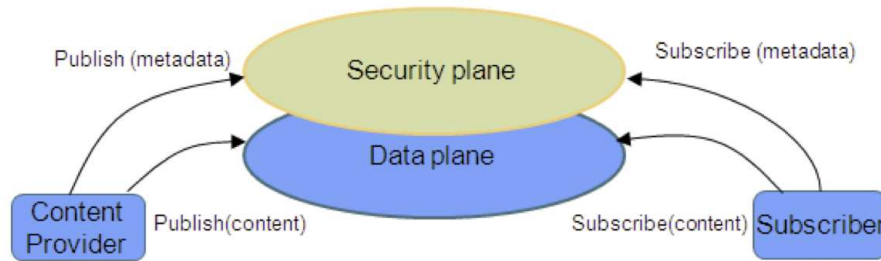


Fig. 4.1: Security plane overview.

The security plane works as a binding point between content providers and consumers. In scenarios where content providers just generate content to be consumed by clients, and there is no security session between them, providers are able to place the security data in the plane to be consumed by clients and satisfy the time decoupling characteristic of these networks. The plane contains *metadata* structures that describe the content itself, together with security parameters needed to authenticate and check the data integrity.

We use the security plane concept together with other information-centric authentication techniques to provide signature amortization in order to efficiently authenticate content pieces and also verify the integrity of the data blocks.

4.2.1 Security Model

The security model separates the content retrieval functionality from the authentication information into different planes: the *security plane*, where the security messages flow and the content descriptors are stored, and the *data plane*, where the data blocks that are part of the content are stored. In order to protect the clients from retrieving fake or bogus content from the data plane, the security plane provides a content descriptor called *metadata*, which contains authentication information about

³The main difference compared to time synchronization regards the fact that the data is produced and stored in some mirror server to be asynchronously consumed.

the content. Fig. 4.2 illustrates an example of metadata proposal to satisfy the basic security model. Further fields can be added according to the application requirements.

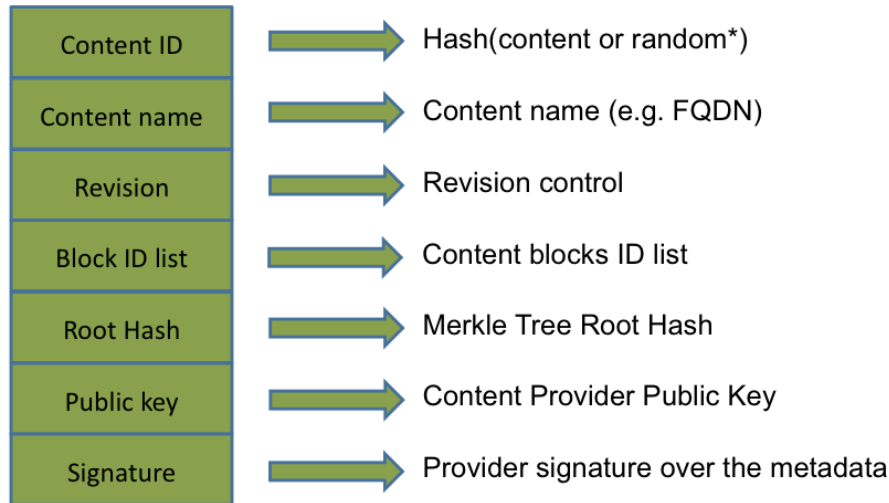


Fig. 4.2: Metadata description.

The *content ID* field contains a randomly generated 256-bit identifier used to identify the metadata in the security plane. The *content name* field contains the high-level content description, for example, a URL for a content. The *revision* field is used to control the current content version for update purposes, e.g., a client may want to receive updates about the *content ID* that she has subscribed for⁴. Thus, the applications need to know how to address different versions of the same content. The *block ID list* contains a sequence of chunks identifiers that will be used by the application to retrieve data from the network.

The *root hash* field contains the root hash of a hash tree, either a *skewed hash tree* or a *composite hash tree*, to be described in the next sections. The root hash is a *fingerprint* over a set of data chunks and allows for data verification without a pre-established security session. The *public key* field contains the public key of the content provider or of a proxy, in case the content provider wants to have anonymity. The *signature* field contains the provider's signature over the metadata to protect against unauthorized modifications on the content descriptor.

In order to insert a metadata in the security plane, content providers must first authenticate in the security plane through public key challenge-response procedure similar to the one used in IDSEC to verify the ownership of the correspondent private key and also a cost function to avoid denial-of-service attacks. After successful authentication, the metadata is inserted in the security plane with a lifetime associated. The security plane can be either public or private. In the first mode, registered content providers can insert metadata in the plane, while in the private mode, clients must authenticate in the plane prior to getting access to the content metadata.

The initial trust between content providers and consumers can be performed in different ways, for

⁴A common example is the case of a client that subscribes to a content, e.g., BBC News, and wants to receive further news updates. However, the BBC News *identifier* can not be changed, otherwise the user will need to refresh the identifier every time it receives an update. Moreover, the lack of permanent identifiers lead to broken links scenarios.

example, local configuration based on administrative rules, integrated with name resolution system or be a totally external entity, such as a distributed hash table.

In the first scenario, the bootstrapping procedure is done by an external administrator, so the local host has the public key embedded in the system, for instance, a list of root certificates in a Web-browser. The benefit of this approach is the pre-establishment of a trust relationship between the software vendor (the entity that embedded the certificates) and the entities that may provide security services (Web-sites that are trusted).

In the second scenario, clients rely on a trusted resolution infrastructure, e.g., DNSSEC. In this approach, users query a distributed system to reach the authoritative server hosting a content provider's name to public key mapping. The benefit of this approach is that key management is easier from the infrastructure point of view, for example, key revocation or key rollover. However, the drawback is the unnecessary trust placed in the resolution infrastructure to resolve queries correctly. For example, if a client trusts her bank (and its digital certificate), then the user does not need to trust in the infrastructure (DNSSEC) to store the bank's public key or resolve the mapping to the bank. The client can spot any forgery because she has the bank's identifier and can check the identity against the bank's certificate.

Finally, the third approach uses a distributed database to store the mappings, such as a DHT. The DHT works as a distributed directory where entities post and request certificates in the Internet. The benefit of this approach is that clients do not need to trust the infrastructure since it is a mere placeholder for the certificates. The trust in the infrastructure is minimum because clients just need to trust that the DHT will store and return the certificates, and clients are able to verify whether a returned certificate is valid or not by checking its digital signature in the returned certificate. Hence, clients do not need to trust the infrastructure to return the correcting mapping, but rather to just return the content provider's digital certificate.

The proposed mechanism is totally location, time and synchronization decoupled. Content providers produce data, generate metadata structures for the data and publish them in the security place. Later, the content providers place copies of the data in strategic places in the network, for example, mirror servers and caches. Clients requesting for the data will need to authenticate in the security place prior to the metadata retrieval. In this scenario, the security plane's server is always active, thus, it is possible for clients to connect to it and authenticate using synchronous protocols, such as IDSEC.

4.3 Information-centric Authentication Mechanisms

This section presents two techniques for data authentication in information-centric networks: *skewed hash tree* and *composite hash tree*. The skewed hash tree is an extension of the Merkle Tree to support random size file authentication, and the composite hash tree is a new hash tree algorithm to authenticate data with configurable overhead.

4.3.1 Skewed Hash Tree

The skewed hash tree mechanism provides random size data authentication with one digital signature regardless of the number of data blocks. The skewed tree extends the original Merkle tree

algorithms to allow data authentication in cases where the number of chunks (data fragments) is not multiple of power of two. As present in Chapter 2, Merkle trees require balanced binary trees to work, thus, it needs some extensions to the algorithms to support unbalanced trees. In order to achieve this requirement, we separate the hash tree into two parts: one balanced tree and a second one with the *skewed leaves*. A skewed leaf is a leaf that is going to be appended under a balanced leaf and it has a special handling in the algorithm. The balanced tree is created over a partitioned content and later the skewed leaves are added under the balanced tree, creating one extra height in the skewed hash tree. The advantage of splitting the tree in balanced tree and skewed leaves is to maintain the compatibility with the original Merkle tree algorithms for the balanced tree while handling correctly the skewed leaves.

Fig. 4.3 illustrates an example of skewed hash tree, where the balanced tree comprehends the leaves with hash values H_{01} , H_{23} , H_4 and H_5 and the skewed leaves contain the hash values H_0 , H_1 , H_2 and H_3 . The skewed tree construction starts with the computation of the smallest tree height that can hold all data blocks minus one⁵, which in this case is $h = 2$ and results in four balanced leaves. Next, the mechanism computes the number of balanced leaves that will receive the skewed leaves in order to hold all data blocks. Finally, it computes the root hash over the data set.

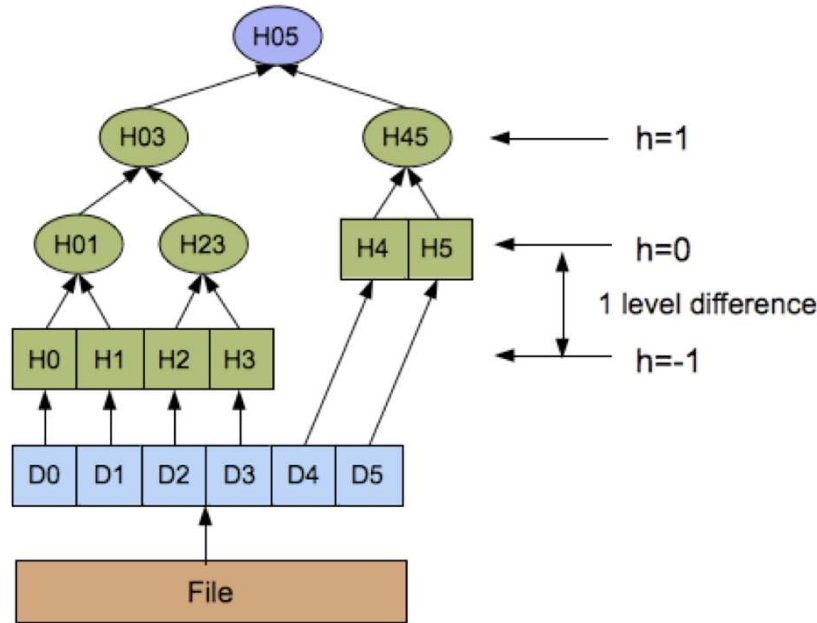


Fig. 4.3: Skewed Merkle Tree with 6 data blocks and one appended level in the skewed hash tree with height 2.

In order to differentiate the skewed leaves from the balanced ones, the skewed leaves are inserted at the height $h = -1$, indicating that they are appended leaves and they should be handled as a special case when using regular Merkle tree algorithms.

⁵The motivation to reduce the tree height in one is to avoid empty leaves, for example, if we choose a tree of height $h = 3$ for this example, we would have 6 data blocks and two empty blocks.

Fig. 4.4(a) illustrates another example of skewed hash tree. In this example, the algorithm computes that there are 6 skewed leaves ($H_{0..5}$) that will be used to generate the balanced leaves in the next height, containing the hash values H_{01} , H_{23} and H_{45} .

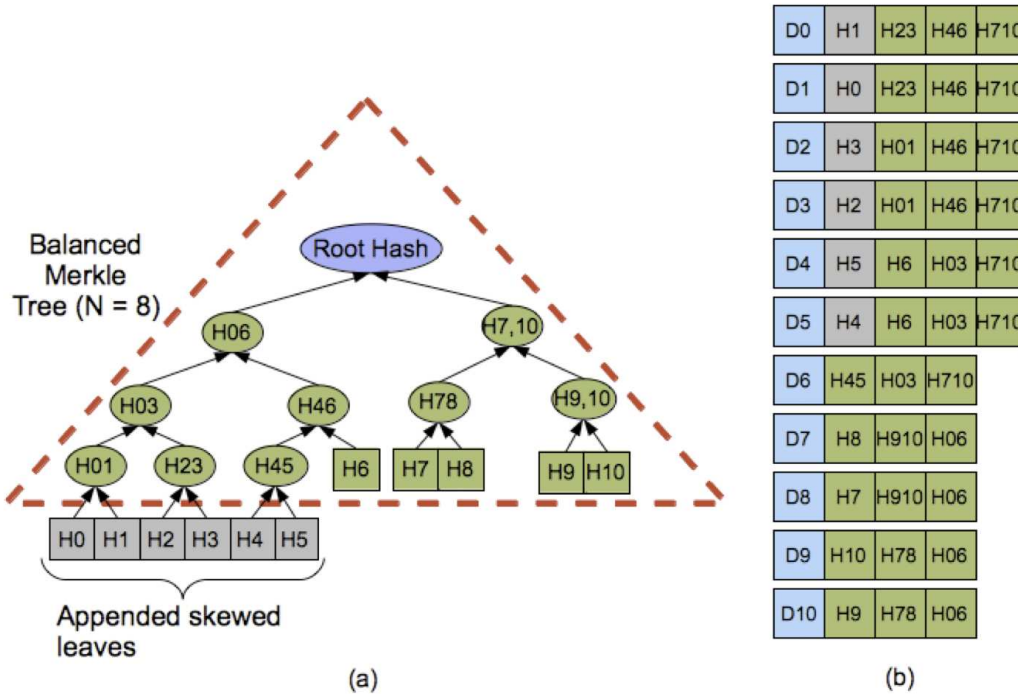


Fig. 4.4: (a) Skewed Merkle Tree with 11 leaves and the inner balanced tree with 8 leaves. (b) Authentication Path for each data block with different lengths.

The algorithm to calculate the root hash starts in the first leaf of the balanced tree, in this case, H_{01} . The first step of the algorithm is to check whether it has skewed leaves appended in that leaf or not. In the example, the leaf H_{01} has appended the skewed leaves H_0 and H_1 , thus the algorithm must compute first these two leaves and later the algorithm returns again to the balanced tree. The balanced tree algorithm now goes to the second leaf H_{23} . It checks whether there are appended leaves or not and treats the skewed leaves. From leaf H_{45} onwards, there is no more skewed leaves, thus, the balanced Merkle tree algorithms can work normally.

Fig. 4.4(b) illustrates each data block with its corresponding AP in the skewed tree. Some data blocks will have an AP of one hash value longer than others due to the skewed tree, but it does not interfere in the verification procedure. The authentication path and verification algorithms also need to be modified to support skewed leaves and they will be detailed in the next sections.

Skewed Merkle Tree Algorithms

The skewed hash tree computations are divided into three phases: *root hash generation*, *AP generation* and *data blocks verification*. The first phase generates the public signature of a target file, the second phase generates the AP for each data block and the third phase authenticates each data

block. In the following algorithms, we use the *stack* data structure to ease the algorithm description and understanding. The decision to use a stack is because it can hold the last two values in the top of the stack, easing the comparison process of the last two values. Also, we consider that the stack has the *pop* and *push(element)* primitives, where *pop* removes the top element of the stack and *push* adds an element in the top of the stack. Before starting the algorithm description itself, we provide some definitions to clarify the operations in the skewed hash tree:

- **Block.** A block or data block is a fragment of a larger file and is considered as the *smallest* unity of data used as input of the skewed hash tree algorithms.
- **Leaf.** A leaf is the bottom node of a binary tree. It contains the cryptographic hash value of a data block.
- **Balanced leaf.** A balanced leaf is a leaf of a balanced binary tree. Even though they are leaves, they may have some skewed leaves appended, but they are called balanced leaves to identify the lowest level of a balanced tree. These leaves can be handled using regular Merkle tree algorithms.
- **Skewed leaf.** A skewed leaf is the leaf that is appended under a balanced leaf. It needs special handling in order to generate a coherent root hash value that can be used in the verification process.
- **Height.** The height h is the total height of the entire skewed hash tree, which is the height of a balanced tree if there is no skewed leaf, or the balanced tree plus one if there are skewed leaves.
- **Root hash.** The root hash is the top value in the skewed hash tree and it is used for the authentication of all data blocks.
- **Authentication path.** The authentication path is a list of hash values that is used to verify a given data block. Each element of the authentication path (AP_i) is the sibling value of the computed hash in the skewed hash tree.

The number of skewed leaves in a skewed hash tree with height h is the number of current leaves in the hash tree minus the number of data blocks of a balanced hash tree with height $h - 1$, multiplied by two⁶. Therefore:

$$num_skewed_leaves = 2 \times (N - 2^{balanced_tree_height}) \quad (4.1)$$

where the *balanced_tree_height* is height of the balanced tree. The number of balanced leaves with appended skewed leaves is:

$$num_balanced_leaves = N - 2^{balanced_tree_height} \quad (4.2)$$

We are going to use these numbers in the skewed hash tree algorithms, detailed in the next section.

⁶Because the next height of a binary tree has two times the number of leaves of the previous height.

SHT Algorithms

There are three algorithms associated to SHT: *skewed_treehash*, *skewed_ap* and *skewed_verify*. The *skewed_treehash* computes the root hash of a skewed hash tree; the *skewed_ap* computes the authentication path for each data block; and *skewed_verify* checks whether a data block is consistent with a given root hash or not. We are going to describe each one in detail.

Alg. 1 describes the root hash generation procedure in a skewed hash tree, which is based on the original *treehash* Merkle tree algorithm presented in [19].

Algorithm 1 SHT treehash algorithm

Input: File, max_height, num_skewed_leaves
Output: Root hash
skewed_count = 0
height = 0
while height <= max_height **do**
 if top 2 values have equal height **then**
 $h_R \leftarrow pop()$
 $h_L \leftarrow pop()$
 height = $h_L.height$
 $h_x \leftarrow hash(h_L || h_R)$
 stack.push(h_x , height + 1)
 else
 if read_data NOT EOF **then**
 data = read_data(file)
 if skewed_count < num_skewed_leaves **then**
 stack.push(hash(data), height=-1)
 skewed_count = skewed_count + 1
 else
 stack.push(hash(data), height=0)
 end if
 end if
 end if
 height $\leftarrow stack[0].height$
end while
Return stack[0]

The algorithm receives as input a file, the block size, the maximum height of the tree (which is calculated dividing the file size by data block size and verifying the smallest height of a balanced tree that can hold that number of leaves) and the number of skewed leaves computed with Eq. 4.1. The algorithm starts reading the data blocks from the file passed as parameter and checks whether the block will become a skewed leaf or a regular leaf in the balanced tree. In the former case, the algorithm pushes the hash of the data block into the stack with height $h = -1$ to indicate that it is a skewed leaf, while in the later case it will have height $h = 0$ to indicate that it is a balanced leaf. Whenever there are two hash values with the same height, the algorithm *pops* those values,

concatenates the popped values and pushes the hash of the concatenation into the stack, incrementing the height of the pushed node. This procedure continues until it reaches the maximum height and the stack will have the root hash value in the top of the stack.

The second phase corresponds to the AP generation for each data block and is divided into two steps: (1) initial stack filling and (2) AP generation. The first step uses the skewed treehash algorithm to store all hash values of the leftmost and rightmost leaves ($h_L \leftarrow pop()$ and $h_R \leftarrow pop()$ in Alg. 1) in the S_h and AP_h stacks respectively. The S_h stack contains the hash value to be used in the next AP generation and the AP_h stack contains the AP value at the height h and it contains the authentication path of the first block. These stacks are used as temporary variables to store the previous hash computed hash values to be used in the next AP computation.

The second step uses the pre-filled S_h and AP_h stacks to output each AP in sequence with one tree traversal. Alg. 2 describes the skewed hash tree traversal algorithm. The algorithm receives as input the file, the number of balanced leaves with appended skewed leaves and the height of the balanced tree and outputs the AP for each data block in sequence.

Algorithm 2 SHT authentication path generation

Input: File, num_balanced_leaves, H
Output: Data blocks with *Authentication Paths*
leaf = 0
skewed_count = 0
if leaf < $2^H - 1$ **then**
 if skewed_count < num_balanced_leaves **then**
 $data_0 = read_block()$
 $data_1 = read_block()$
 Output $data_0$, hash($data_1$), AP
 Output $data_1$, hash($data_0$), AP
 skewed_count = skewed_count + 1
 else
 data = read_block()
 Output data, AP
 end if
 for $h = 0$ to H **do**
 if (leaf + 1) mod $2^h == 0$ **then**
 $AP_h = Stack_h$
 startnode = (leaf + 1 + 2^h) XOR 2^h
 $Stack_h = skewed_tree_hash(startnode, h)$
 end if
 end for
 leaf = leaf + 1
end if

Fig. 4.5 shows an example of the algorithm working on the skewed hash tree presented in Fig. 4.3.

In Fig. 4.5(a), a skewed hash tree is constructed over six data blocks, resulting in a balanced

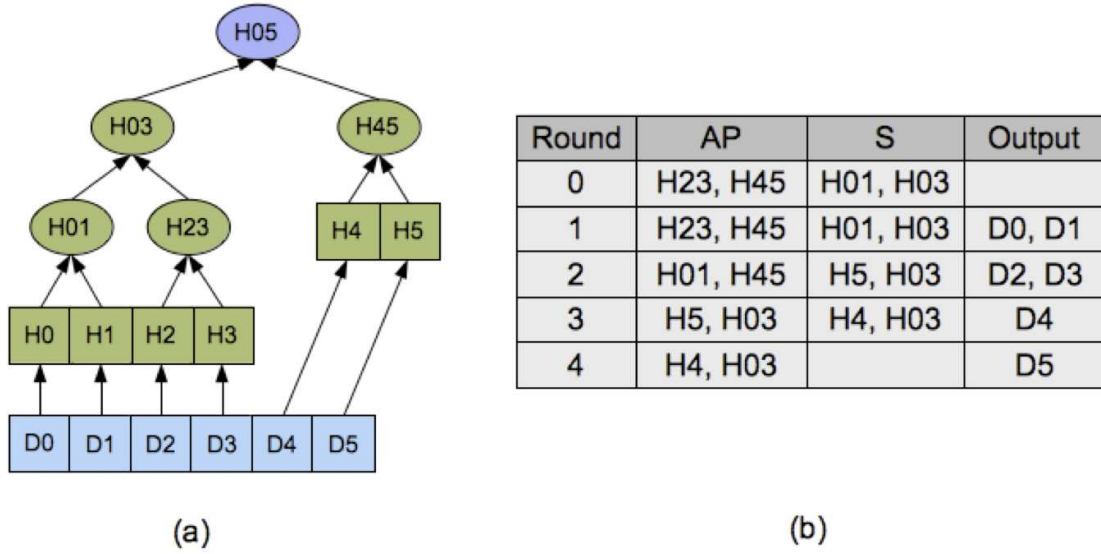


Fig. 4.5: (a) Skewed hash tree with $h = 2$. (b) Table containing the output on each round of the algorithm.

Merkle Tree with $h = 2$ and four skewed leaves. Table 4.5(b) shows the output of the algorithm on each round. The first step of the AP generation is performed in the round zero. Thus, we have the AP_h and S_h stacks initially filled with $AP_0 = H_{23}$; $AP_1 = H_{45}$ and $S_0 = H_{01}$; $S_1 = H_{03}$ respectively (all left node values in the S_h stack and all right node values in the AP_h stack). In the first round, the algorithm verifies if there is any skewed leaf appended under the balanced leaves. In our example, there are two skewed leaves, thus, the algorithm computes the hash values of each data block and outputs the blocks D_0 and D_1 with their respective sibling hash values (H_1 and H_0 respectively) and the APs in the AP stack (H_{23} , H_{45}). The algorithm fills the AP stack with new hash values and goes to the second leaf of the balanced tree (H_{23}), performing the same procedure as the first step. In the third and forth leaves, the algorithm outputs one data block with their correspondent AP since there is no more skewed leaves.

The third phase comprehends the data block verification procedure, described in Alg. 3, where the receiver gets the data block with its corresponding AP and the block index. We assume the root hash was previously transferred to the receiver in a secure way, for example, using the security plane model. The algorithm starts reading the data block's AP and appends each hash value in the correct side to reach the root hash. One important notice is that the AP at each height should be appended in the correct position (left or right) using the block index since during the creation they also follow a correct order ($hash(A || B)$ is different to $hash(B || A)$) and will result in a different hash value.

The block index represents the path from the leaves up to the root of the skewed tree. For example, in the Fig. 4.5, the block indexes for data blocks $D_{0..5}$ are 0, 1, 2, 3, 3, 4 and they are used together with the authentication path length to verify a data block. The block index does not represent the sequence order of data blocks, but the topological location in the tree to perform the authentication using the APs.

Algorithm 3 SHT verification

Input: Root Hash, block index, data block, AP
Output: True or False

```

pos = index
digest = hash(data_block)
for each  $AP_i$  value in  $AP$  do
    if (pos % 2 == 0) then
        digest = hash(digest ||  $AP_i$ )
    else
        digest = hash( $AP_i$  || digest)
        pos =  $\lfloor pos/2 \rfloor$ 
    end if
end for
if (digest == Root Hash) then
    Return True
else
    Return False
end if

```

4.3.2 Composite Hash Tree

This section presents the Composite Hash Tree (CHT), a data structure containing summary information of partitioned piece of data which can be authenticated with one digital signature. First, we will introduce the CHT data structure and describe how CHT can be tuned using the h and α parameters and also present some authentication example. Second, we will present the proof of the authentication overhead complexity, discussing the advantages and limitations of the CHT approach compared to SHT and Merkle trees. Third, we will present the algorithms used to construct and authenticate the CHT.

CHT Overview

The Composite Hash Tree (CHT) is a data structure created over a set of data blocks belonging to a complete file. The main idea is to create a set of small binary hash trees of fixed height over a set of data blocks and recursively construct other binary hash tree over the previous hash trees in the first level, until reaching one single hash tree in the top level. The motivation for this approach is the high overhead present in the Merkle tree and also skewed hash tree, because the latter one is mainly based on the original Merkle tree algorithms. In these approaches, each data block has a list of cryptographic hash values (authentication path) that is the same length of the hash tree. Therefore, each authentication path has $\log_2 N$ values and the sum of all authentication overhead grows $N \times \log_2 N$, where N is the number of blocks. Thus, for large files, this overhead might be considerable, especially in scenarios using low processing devices such as mobile phones.

In order to attack the authentication overhead problem, we propose CHT as an alternative to both Merkle and skewed hash trees for authentication purposes with low overhead. The proposed mechanism also provides signature amortization, allowing one piece of content to be authenticated with one

digital signature regardless of the number of data blocks, requiring on average $O(N)$ fingerprints to authenticate N data blocks that are components of the original content for small composing Merkle tree with height h . In order to better describe the CHT data structure and the verification procedures associated to it, we start with some definitions used through the text to ease the comprehension of the proposed mechanism. Although many of the concepts are similar to the ones presented in the skewed hash tree section, we added them here to keep the section self-contained.

- **Hash Tree (HT).** A binary hash tree [19] is a complete binary tree with height h and 2^h leaves. Each leaf stores a cryptographic hash value of over a data block and each internal node stores the hash of the concatenation of its children's node
- **Root Hash (RH).** The *Root Hash* is the hash value in the top of an intermediate hash tree, representing the signature over a set of data blocks. The RH algorithmically binds together all data blocks, and any change in any data block will result in a different signature.
- **Composite Root Hash (CH).** The *Composite Root Hash* is the hash value in the top of a composite hash tree used to authenticate the incoming *Authentication Data Blocks*. The CH can be digitally signed to provide both content authentication and integrity regardless of the number of data blocks.
- **Authentication Data Block (AD).** The *Authentication Data Block* contains intermediate RH values of the hash trees used in the composition. It is used to authenticate the smaller trees and data blocks as they arrive in the receiver side.
- **Authentication Path (AP).** The *Authentication Path* is the list of hash values needed to authenticate a specific data block. The AP hash value in a given height h is the sibling hash in the hash tree towards the root hash. The main difference between AP and AD is that the first one is used to authenticate one data block and the second one is used to authenticate the RH of intermediate hash trees.

A $\text{CHT}(\alpha, h)$ is a composite hash tree using smaller Merkle trees of height h ($\text{MT}(h)$) whose root hash values are aggregated in blocks of α elements, where $\alpha \geq 2$ (The motivation for $\alpha \geq 2$ is because for $\alpha = 1$ the cryptographic hash function will be applied two times over the same data block without any real security benefits. For $\alpha = 2$, it works as a regular Merkle Tree). Fig. 4.6 illustrates an example of $\text{CHT}(2, 1)$ using internal hash tree value $h = 1$ and intermediate RH aggregation of two blocks ($\alpha = 2$). In this example, a file is divided in eight data blocks (D_0 to D_7) and an intermediate hash tree of height $h = 1$ is constructed using the cryptographic hash of the data blocks as input (H_0 and H_1), resulting in an intermediate *Root Hash* (H_{01}). This intermediate RH is used as the verification information for the data blocks D_0 and D_1 , which later on will be aggregated in *Authentication Data Blocks*.

The CHT has two configuration parameters: aggregation index (α) and internal hash tree height (h). The α parameter is used to define the aggregation level of the intermediate RH values in the binary hash tree in α values. The internal hash tree height (h) defines the height of the internal hash trees used in the composition. These two parameters allow for the customization of the tree behavior, for instance, the initial verification ordering and the verification overhead, according to the application requirements. High h values provide smaller authentication hierarchy, meaning that data

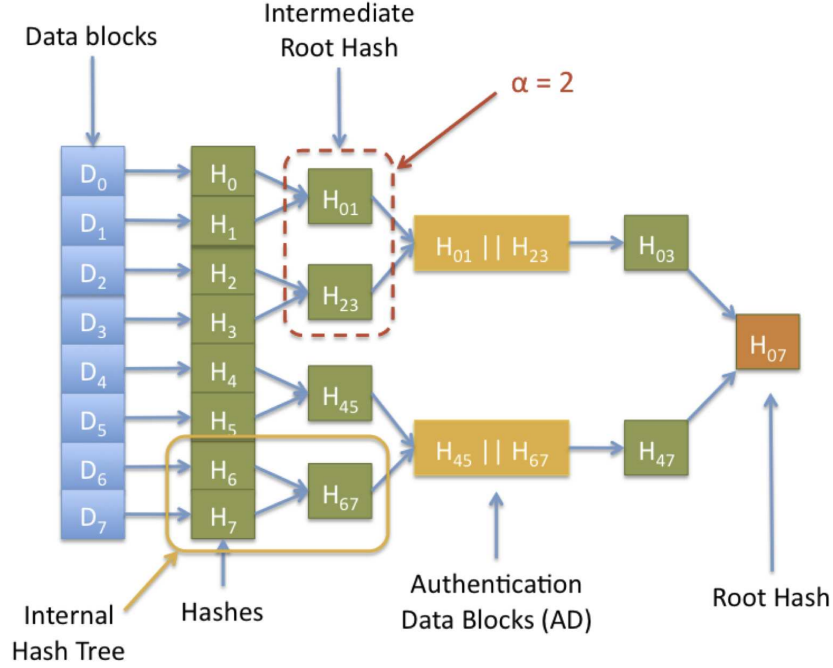


Fig. 4.6: (a) Composite Hash Tree with internal HT of height $h = 1$ and $\alpha = 2$.

and authentication blocks have low interdependency at the cost of higher authentication overhead per data block. On the other hand, small h values results in low authentication overhead, but longer data block authentication hierarchies (thus, higher dependency between data and authentication blocks). High h values result in fewer number of internal Merkle trees because a $MT(h = 2)$ has the double number of leaves than a $MT(h = 1)$. However, the AP grows with the height of the Merkle Tree, resulting in higher authentication overhead.

In this example of Fig. 4.6, intermediate RH values in the first level (H_{01} and H_{23}) are aggregated together in blocks of two ($\alpha = 2$), resulting in the *Authentication Data Blocks* with hash values $H_{01} || H_{23}$ and $H_{45} || H_{67}$, where $||$ represents the concatenation operation. In the second level, the *Authentication Data Blocks* are considered as input data blocks. Hence, the CHT applies the cryptographic hash over the ADs, resulting in the hash values H_{03} and H_{47} and another intermediate hash tree of height $h = 1$ is constructed over these two data blocks, resulting in the *Composite Root Hash* which will be used in the verification procedure. In case of larger files, this procedure is applied recursively until reaching the *Composite Root Hash*.

In order to provide data verification, each data chunk carries a list of hash values represented by the AP used to verify with the CH. The AP for each data block is the sibling hash value in the hash tree, for instance, in the example described in Fig. 4.6, the AP for the D_0 is H_1 , since this value is the sibling value of H_0 . For larger hash trees, the AP is composed of the sibling hash value at each height towards the RH⁷. Therefore, the overhead per data chunk is defined by the height of the internal hash tree. In this approach, the CHT maintains just one hash value needed to authenticate a target data

⁷Recalling that the AP length is the height of the Merkle Tree, thus, this is the motivation to use really small Merkle trees.

block, discarding the repeated values of the regular Merkle Tree, as described in Chapter 2. On the other hand, this mechanism introduces an authentication hierarchy between data and authentication blocks, requiring that some blocks to be authenticated prior to the data blocks authentication.

As described in the examples above, the CHT mechanism trades authentication overhead (length of the authentication path) with the size of the authentication hierarchy. In order to make the CHT more flexible, we introduce the *root hash aggregation index* α . The index sets the number of root hash values that must be aggregated together before sending it to the Merkle Tree of the next level. Thus, rather than passing one root hash at a time to the next level, the mechanism aggregates α root hashes to create the *authentication data block* (AD) to be passed to the next level. Hence, we can transfer a bulk of root hash values and reduce the total height of the CHT. The α index reduces the authentication hierarchy needed to authenticate all data blocks in an order of α elements. Thus, the index reduces $\log_{\alpha} N$ authentication levels, where N is the number of partitioned data blocks.

Fig. 4.7 illustrates an example of authentication hierarchy using a sliding window for a CHT(2, 1). The figure has two columns, the first one indicates the received data blocks in the receiver side and the second column shows the next blocks window to be downloaded next. As authentication blocks arrive, the next blocks to be downloaded *slides* to the next set of data block that can be downloaded with the arrival of the new set of *Root Hashes*. For example, after the receiver authenticates the AD_0 containing the hash values $H_{01} \parallel H_{23}$, the user can start downloading data blocks D_0, D_1, D_2 and D_3 , in any sequence.

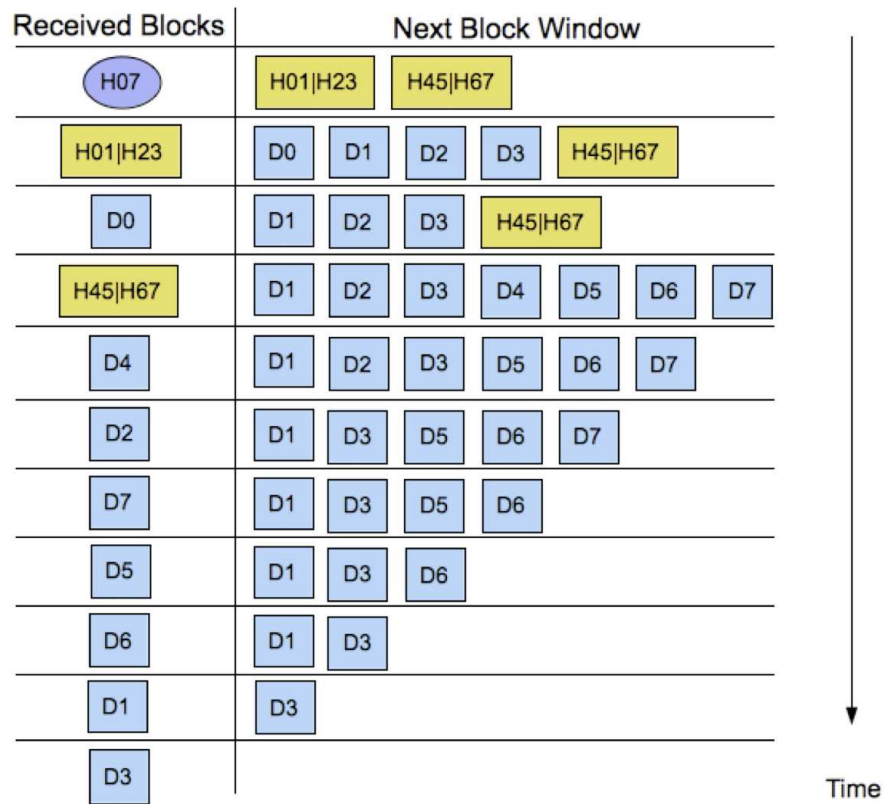


Fig. 4.7: Authentication Window for CHT.

The same procedure is taken when the AD with concatenated hash values $H_{45} \parallel H_{67}$ is received in the destination, allowing the download and authentication of data blocks D_4, D_5, D_6, D_7 in any sequence.

Fig. 4.8 shows a $CHT(4, 1)$ with height $H = 2$ for 16 data blocks. The α index allowed the construction of a CHT with the same data block authentication overhead of a $CHT(X, 1)$ (one hash value per data block and X represents any α value) while kept the same number of authentication hierarchies as a $CHT(2, 1)$ (one authentication hierarchy). Thus, higher α values minimizes the number of authentication hierarchies in the CHT and this option is interesting for applications that require low verification overhead and they do not have problems with authentication dependency.

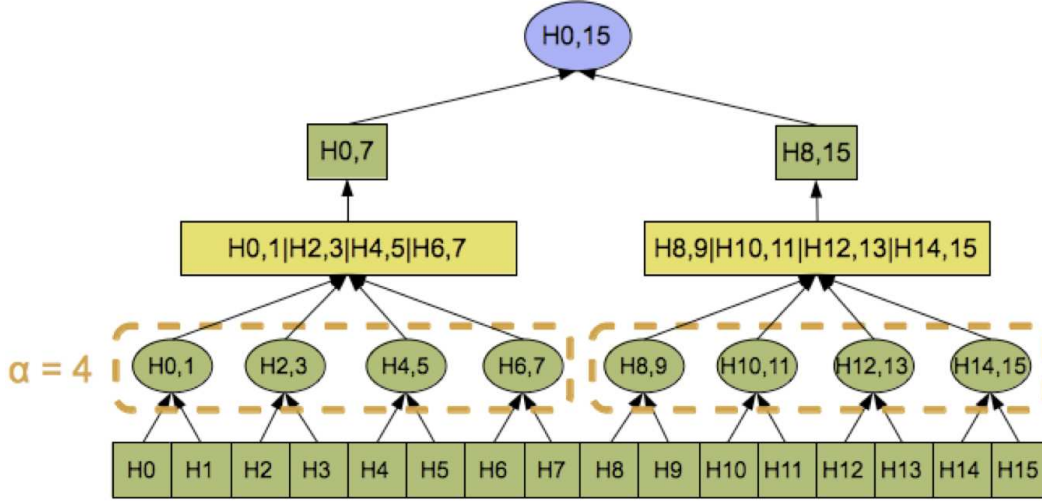


Fig. 4.8: $CHT(4, 1)$ for 16 data blocks with $H = 2$.

CHT Overhead Complexity

The CHT overhead has two components associated, the *Authentication Path* (O_{AP}) overhead of each data and *Authentication Data Block* (O_{AD}) overhead, which are the aggregated *Root Hash* values of the intermediate Merkle Trees. Thus, the total overhead is:

$$O_T = O_{AP} + O_{AD} \quad (4.3)$$

The O_{AP} is the sum of the product between the number of data blocks on each height by the size of the AP, which is defined by the height of the Merkle Tree used in the CHT. From the CHT construction example above (Fig. 4.8), we can notice that the factor $2^h \alpha$ repeats recursively i times to create the CHT over the data blocks. Note that the last MT created over the data blocks does not follow the pattern because they are the data blocks on which the composite hash tree is being created over. These data blocks add 2^h leaves, thus we need to add it separately in the formula to compute the overhead. Therefore, the O_{AP} formula is the product of the i recursions plus 2^h leaves over the data blocks plus the AP length (which is the same as h).

$$O_{AP} = \sum_{i=0}^{H'} (2^h \alpha)^i \times 2^h \times (AP \text{ length} = h) \quad (4.4)$$

where H' represents the CHT height minus 1⁸. The H' is the number of data blocks N minus the $MT(2^h)$ over the data blocks that do not repeat. Therefore:

$$H' = \lceil \log_{(2^h \alpha)} (N/2^h) \rceil \quad (4.5)$$

The O_{AD} is similar to the AP overhead formula and computes the sum of the product of the intermediate *Root Hash* values that are aggregated into α hash values, excluding the $MT(2^h)$ over the data blocks since it starts from the first level. Hence:

$$O_{AD} = \sum_{i=1}^{H'} (2^h \alpha)^i \times (AP \text{ length} = h) \quad (4.6)$$

In order to calculate the overhead complexity with the input, we first calculate the total number of data blocks of a $CHT(\alpha, h)$. From 4.5, we have that:

$$N = (2^h \alpha)^{H'} \times 2^h \quad (4.7)$$

From 4.4 and substituting with 4.7, we have:

$$O_{AP} = \sum_{i=0}^{H'} (2^h \alpha)^i \times 2^h \times h \approx (2^h \alpha)^{H'} \times 2^h \times h = N \times h \quad (4.8)$$

Therefore the O_{AP} in the CHT is $N \times h$ and grows $O(N)$ when $N \gg h$ and h is a constant that does not change with the input size. The maximum value for h in a binary tree is $\log_2 N$, reducing the CHT to a regular Merkle Tree with overhead complexity of $O(N \log_2 N)$.

The *Authentication Data Block* overhead has similar proof to the previous one. Thus, substituting in 4.6, we have:

$$O_{AD} = \sum_{i=1}^{H'} (2^h \alpha)^i \times h \approx (2^h \alpha)^{H'} \times h = (N \times h)/2^h \quad (4.9)$$

Therefore, the O_{AD} in the CHT is $N \times h/2^h$ and grows $O(N)$ when $N \gg h$ and h is a constant parameter that does not change with the input size. The total CHT overhead (O_T) is:

$$O_T = N \times h + (N \times h)/2^h = O(N) \quad (4.10)$$

⁸It is not considered the *Root Hash* in the height computation (thus, $H' = H - 1$).

Table 4.1 compares the overhead of a regular Merkle Tree and a CHT(1, 2):

Tab. 4.1: Merkle Tree vs. Composite Hash Tree Overhead Comparison

# of blocks	Merkle Tree	Composite Hash Tree (1,2)	Overhead Reduction(%)
8	24	12	50.00
32	160	48	70.00
128	896	192	78.57
512	4,608	768	83.34
2,048	22,528	3,072	86.36
8,192	106,496	12,288	88.46
32,768	491,520	49,152	90.00
131,072	2,228,224	196,608	91.18
524,288	9,961,472	786,432	92.10

The MT above presents an overhead of $O(N \log_2 N)$ while the CHT(1, 2) has an overhead of $N \times h + (N \times h)/2^h = 2N + N = 3N = O(N)$. Thus, the difference between the Merkle Tree and the Composite Merkle Tree grows $O(\log_2 N)$ with the input. The reduction of the CHT authentication overhead compared to a regular MT is due to the hierarchical verification procedure required in the CHT.

CHT Algorithms

The CHT construction and verification has three phases: CHT root hash generation, *authentication data* generation and data blocks verification. The first phase comprehends the computation of the *root hash* that will be used to authenticate the partitioned data by the receiver. The second phase generates the authentication data including the *authentication data blocks* with the root hash values of each smaller Merkle Tree and the AP for each data block. The third phase comprehends the verification procedure of each data block, which requires a hierarchical authentication procedure.

Alg. 4 describes the CHT root hash computation algorithm. It uses a modified version of the Merkle Tree *treehash* algorithm described in [19] and it is used to calculate intermediate root hash values in Merkle Trees.

The algorithm receives as input the *aggregation index* α , the Merkle Tree internal height h used in the composition, the input data file, and the maximum height of the tree, computed using the Eq. 4.5⁹. All algorithms use a *stack* to store the intermediate values.

The CHT root hash generation procedure starts reading data blocks and storing the hash of the blocks in the *hash tree stack* (S_{ht}). The hash tree stack contains the hash values that need to be aggregated by α values. Once upon there are α values in the S_{ht} , the algorithm removes all the top α values, aggregates them and applies a cryptographic hash over the set. Then, it pushes the resulting hash value into the *Merkle tree stack* (S_{mt}). The Merkle tree stack contains all values that need to be cryptographically hashed using the regular Merkle Tree algorithm, adding the resulting hash value in the S_{ht} . This process continues until the CHT root hash is obtained.

⁹The total height is $H = H' + 1$

Algorithm 4 CHT *root hash* generation procedure

Input: $\alpha, h, \text{File}, \text{max_height}$
Output: CHT root hash
height = 0
while height < max_height **do**
 if top α values have equal height (*ht_stack*) **then**
 cur_height = height
 $h_i \leftarrow \text{pop the last } \alpha \text{ values}$
 local_ad_ap $\leftarrow \text{AGGREGATE}(h_i)$
 mt_stack = add(local_ad_ap, cur_height + 1)
 else
 if top h values have equal height (*mt_stack*) **then**
 cur_height = height
 startnode = len(mt_stack) - 2^h
 local_rh = treehash(startnode, h, mt_stack)
 ht_stack = add(local_rh, cur_height + 1)
 else
 local_rh = treehash(startnode, h, mt_stack)
 ht_stack = add(local_rh, 0)
 end if
 end if
end while
Return ht_stack[0]

Algorithm 5 Auxiliary treehash algorithm

Input: startnode, max_height, data_source
Output: local root hash
height = 0
while height < max_height **do**
 if top 2 values have equal height (*stack*) **then**
 cur_height = height
 $h_R \leftarrow \text{pop}()$
 $h_L \leftarrow \text{pop}()$
 local_rh $\leftarrow \text{hash}(h_L || h_R)$
 stack = add(local_rh, cur_height + 1)
 else
 Read data_block
 local_hash $\leftarrow \text{hash}(\text{data})$
 ht_stack = add(local_hash, 0)
 end if
end while
Return ht_stack[0]

The algorithms use one auxiliary procedure named *treehash*, which calculates the root hash of smaller hash trees (described in Alg. 5). It receives as parameters the starting node, i.e., the data block index or the stack index, the height to which it should compute and the source of the data, represented by a stack. In this case, if the function is called with *treehash*(0, 2, *mt_stack*), it will start reading the node starting with index zero in the *mt_stack* and applying the Merkle tree algorithms until reaching the height of $h = 2$. In this example, it will read four blocks ($N = 4$) because it reads from a binary tree (2^h , where $h = 2$, thus, $2^2 = 4$).

The second CHT algorithm describes the tree traversal mechanism to generate the AP for each data block together with the corresponding authentication data blocks. Alg. 6 works similarly to the previous CHT root hash generation algorithm but it also stores the authentication path data. It receives the α and h parameters, the input file and the maximum height for the tree. While the algorithm calculates the root hash of the CHT tree, it stores the intermediate AD and the AD's AP blocks (or outputs for external storage) to be used by any receiver. For the data blocks, we use the skewed hash tree authentication path generation, described in Alg. 2.

Algorithm 6 CHT root hash generation procedure

Input: $\alpha, h, \text{File}, \text{max_height}$
Output: CHT root hash
height = 0
while height < max_height **do**
 if top α values have equal height (*ht_stack*) **then**
 cur_height = height
 $h_i \leftarrow \text{pop the last } \alpha \text{ values}$
 Output h_i outputs AD_i *
 local_ad_ap $\leftarrow \text{AGGREGATE}(h_i)$
 Output $\text{AD}_{\text{local_ad_ap}}$ outputs the AP_i for AD_i *
 $\text{mt_stack} = \text{add}(\text{local_ad_ap}, \text{cur_height} + 1)$
 else
 if top h values have equal height (*mt_stack*) **then**
 cur_height = height
 startnode = len(*mt_stack*) - 2^h
 local_rh = *treehash*(startnode, h , *mt_stack*)
 $\text{ht_stack} = \text{add}(\text{local_rh}, \text{cur_height} + 1)$
 else
 local_rh = *treehash*(startnode, h , *mt_stack*)
 $\text{ht_stack} = \text{add}(\text{local_rh}, 0)$
 end if
 end if
end while
Return $\text{ht_stack}[0]$

The verification procedure algorithm is described in Alg. 7, which requires a hierarchical authentication procedure. The tree at height $H - 1$ is required to authenticate the tree at height $H - 2$, until it gets to the root hash of the tree just above the data block, as illustrated in Fig. 4.7. The algorithm

receives as input the CHT root hash and the authentication data blocks with their authentication paths. It also uses two stacks, S_{ht} and S_{mt} , to store the intermediate hash values. The algorithm starts reading the AD blocks together with their APs and inserts the resulting hash value into the S_{ht} until there are α values. Then, the algorithm removes the last α values and adds them in the S_{mt} stack to be added using the Merkle Tree algorithm, repeating until the CHT root hash is calculated. Then, the algorithm compares the calculated root hash with the given one, returning when if they are equal or not (**true** or **false**). For the data blocks, we use the data block verification mechanism described in Alg. 3.

Algorithm 7 CHT Verification Procedure

Input: $\alpha, h, \text{File}, \text{max_height}, \text{root_hash}$
Output: True or False
 height = 0
while height < max_height **do**
 if top α values have equal height (ht_stack) **then**
 cur_height = height
 $h_i \leftarrow \text{pop the last } \alpha \text{ values}$
 local_ad_ap $\leftarrow \text{AGGREGATE}(h_i)$
 $mt_stack = \text{add}(\text{local_ad_ap}, \text{cur_height} + 1)$
 else
 if top h values have equal height (mt_stack) **then**
 cur_height = height
 startnode = len(mt_stack) - 2^h
 local_rh = treehash(startnode, h, mt_stack)
 $ht_stack = \text{add}(\text{local_rh}, \text{cur_height} + 1)$
 else
 for $i = 0$ to h **do**
 $h_i \leftarrow \text{ad_list}_i$
 end for
 $h = \sum h_i$
 $mt_stack = \text{add}(h, 0)$
 end if
 end if
end while
if $ht_stack == \text{root_hash}$ **then**
 Return True
else
 Return False
end if

One scenario that is also considered is the unbalanced CHT, where the number of data blocks is not multiple of $2^{h+1}\alpha$. In this case, the CHT algorithms must have appended the skewed hash algorithms to deal with this special case. Similar to the skewed tree case, the skewed leaves are appended under the balanced CHT leaves and they are handled before going to the regular CHT algorithm.

4.4 Summary

This chapter has presented content authentication mechanisms in information-centric networks. We have described the initial proposal of a security plane to separate the data retrieval procedure from the content authentication as a requirement for information-centric networking (ICN). Later, we have proposed two authentication mechanisms for data verification using hash trees, named skewed hash tree and composite hash tree. The proposed mechanism focus on content authentication regardless of the data source, establishing explicit trust with the original content provider.

Capítulo 5

Application Scenarios for Information-centric Data Authentication

This Chapter presents the application of the skewed and composite hash trees techniques in different scenarios. We start with the secure content name resolution to the metadata as the initial step for the data retrieval process. Next, we present two application scenarios for the composite hash tree over legacy information-centric applications, namely, pollution detection in Peer-to-peer networks and parallel authentication over HTTP. Finally, we propose the usage of hash tree mechanism for secure content caching in the network.

5.1 Towards Secure Name Resolution

This section proposes an information-centric naming system to bind content names and data blocks in a secure way [21, 22]. We start with the motivation of our work and describe how hash tree techniques can provide secure name resolution, trust transfer and content integrity for application data.

5.1.1 Motivation

The Domain Name System (DNS) was introduced in the Internet to overcome the administrative burden caused by the growing number of hosts, mainly due to the manual configuration of the `/etc/hosts` file¹. As more and more computers were deployed around the world, the size of the `/etc/hosts` file grew to unmanageable sizes, demanding a new resolution system that was at the same time scalable, distributed, and administratively easy to manage. The basic functionality required in that context was to resolve a hostname into an IP address, easing the access to remote computers through names rather than memorizing the sequence of unfriendly numbers of an IP address.

Despite the success of the naming system, there are many security flaws in the resolution design, mainly because it was not designed with security in mind. The naming system is prone to a number of attacks, such as denial-of-service attacks, DNS cache poisoning, among others examples[61]. We

¹This file contains a list of hosts and their corresponding mappings to IP addresses and it is present in many Linux distributions.

argue that one of the root causes lies in the *implicit trust* placed by clients on the resolution system. Users *trust* that the resolution system will always return the correct IP address of the destination host or resource. The motivation of our work is to provide a more flexible naming system that binds an *authority* (content provider or owner) and a secure identifier to identify pieces of content in the Internet, allowing for the content verification and directly trust establishment with the content provider rather than placing trust in the resolution mechanism. Additionally, the proposed resolution mechanism is an alternative to resolve high-level content names (e.g. URLs) into the content metadata, which will be used in some application scenarios described in the next section.

5.1.2 Naming System Design

In order to tackle the security issues in the naming system, we propose a set of design goals with *simplicity*, *scalability*, *robustness* and *security* in mind:

- **Backwards compatibility.** The new naming system should be backwards compatible and also be incrementally deployable in the current Internet architecture, thus, allowing its usage in both legacy and pure information-centric networks;
- **Content authentication, identification and location splitting.** Content authentication, identification and location splitting leverage content storage and retrieval from multiple locations, content replication and mobility²;
- **Secure content identification.** Persistent content identification with security properties allows for location-independent labels, making it suitable for caching and also as a homogeneous namespace for content identification;
- **Provenance.** the naming system should provide mechanisms to *verify* and *validate* the content pieces with the original provider. The main idea is to migrate the security from the storage location (mirror server) to the content and the authority over the content.

Additionally to these design goals, we aim at providing the initial resolution step from a single name or URL to a content metadata and later to the block identifiers. In that sense, we aim to transfer the initial trust placed on the name to the metadata, and lastly to the data blocks, ensuring content authentication and integrity, which will be discussed below.

Initial design

In order to design a secure naming scheme, we introduce three definitions, which represent the main components of the naming scheme: *authority*, *content* and *location*.

- **Authority.** Authority is any entity who, in the first place, has the direct control over the data stored and handled by the system. It can be either a content provider who generated the content itself, thus, the content owner, or an appointed entity to act as a representative of the content owner, e.g., a proxy.

²The location splitting requirement is important because content should not be bound to specific locations in information-centric networks.

- **Content.** Content is any resource or piece of information that is generated by an authority, which can be stored in a location.
- **Location.** Location is a position in the memory, hard-disk or network where a piece of content can be stored and located.

Large pieces of content can also be partitioned into smaller data chunks of fixed size to satisfy external requirements, for instance, maximum transmission unit (MTU) in the network. In addition, data can be fragmented due to system policies, e.g., Peer-to-peer networks require content to be divided and exchanged in smaller chunks to increase the overall availability of the system. Therefore, data chunks are smaller components of a large content and the sum of all parts forms a content.

The mapping of these concepts to the network level is represented by the *authority*, *content* and *location identifiers*. The benefits of using these *secure tokens* in the network level are twofold: first, we decouple the *authority* from any specific location in the network, leveraging a broader concept of *authority* rather than a single administrative domain; second, we add security semantics within an identifier, easing the authority authentication in the network. As a consequence, entities are able to recognize a security token that has been confirmed (authenticated) before, regardless of the location, and they are able to recreate an indicative relationship between a message and an already known *authority*.

5.1.3 Implementation architecture

This section proposes an implementation architecture for an information-centric name resolution. The section starts with the resolution procedures to resolve authority and content names into their corresponding network-level identifiers, discussing the trust transfer on each step. Next, we present the options to resolve the content identifier into its corresponding metadata and, lastly, we describe the trust transition from the metadata to the data chunks using hash tree techniques.

Initial approach

We use Universal Resource Identifiers (URIs) [62] as the foundation for our naming scheme due to its backwards compatibility with the current naming system. The URIs introduce a set of names and addresses that is used to identify resources in the Internet, leveraging global search and retrieval of documents across different operating systems and protocols. An URI is mainly composed of three components: a *scheme*, an *authority* and a *resource path*, as illustrated in Fig. 5.1 (additional components are described in [62]).

The *scheme* defines the protocol handler for the *authority*; the *authority* is the owner or the responsible for the *resource* and the *resource path* points to the resource in the *authority* namespace. In the current naming system, an URI is actually mapped to an URL and the *authority* becomes the Fully Qualified Domain Name (FQDN) of the network where the resource is located³. The domain name is resolved into an IP address through DNS and clients are able to request the *resource* to the returned IP

³Of course, there may be additional fields within the authority field. However, as they are not commonly used and not essential for the current discussion, we ignore them in this thesis.

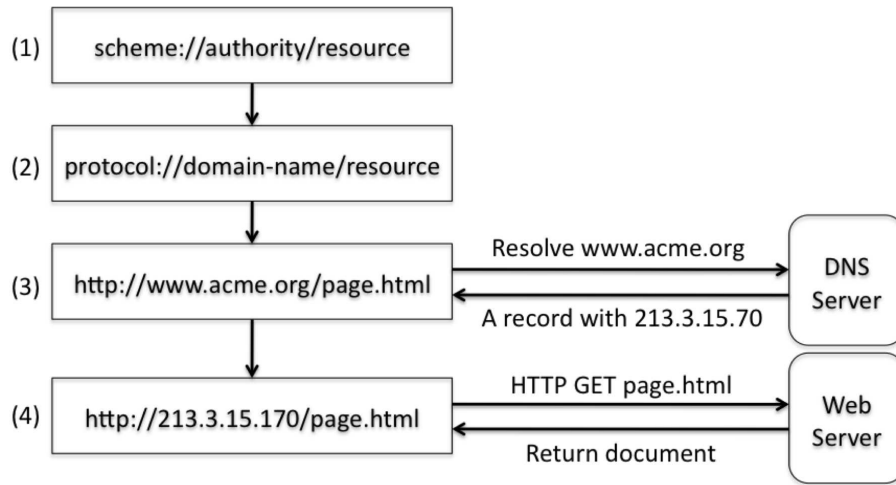


Fig. 5.1: URI naming scheme and resolution in the current DNS.

address. However, the main drawback is the binding between content and a specific, relatively stable set of locations.

We take the stance that the *authority* field defines the *content owner*, or the immediate entity responsible for the content, rather than defining just of a namespace authority. Hence, rather than mapping an FQDN to a location-dependent identifier in the DNS, the resolution system maps the authority field to the public key (or other secure identifier) of the authority over the data, providing a data verification mechanism. In other words, we go from a scenario with coarse grained granularity, e.g., the *authority* field representing a whole administrative domain, to a fine-grained scenario, where the *authority* is used to denote the organization or person responsible for the content the name points to.

Fig. 5.2 illustrates an example of an information-centric name resolution, where the authority and the content names are resolved into an authority and content identifier in a directory service (step 2). We also map the *resource path* to a content identifier which is also location-independent, which will be resolved to a *content metadata*. Later, the content identifier can be resolved into a metadata structure that will be used by applications to retrieve the data blocks (step 3).

Fig. 5.3 illustrates the direct trust establishment between a client and a content provider. In this scenario, a client wants to receive news from the BBC News web-site. Thus, it resolves the authority name (e.g. **bbc.co.uk**) into its corresponding network-level identifier by retrieving BBC's digital certificate and the content identifier. After the digital certificate is validated, the authority identifier can be obtained by applying a cryptographic hash over the public key. Therefore, we transfer the trust from the signer of the digital certificate to the authority identifier because we trust the signer of the digital certificate, e.g., VeriSign (we also discuss other alternatives to establish the initial trust). Later, we resolve the content identifier into a corresponding metadata in a resolution service⁴ to obtain the metadata. As the content provider signs the metadata, we are able to verify its integrity and also *trust* that it is authentic, since we have verified the authority's identifier. The dashed lines in the figure

⁴The decision to separate the content identifier resolution to the metadata in a separate step is to keep a permanent identifier for applications. Otherwise, any change in the metadata structure would change the content identifier, resulting in broken links.

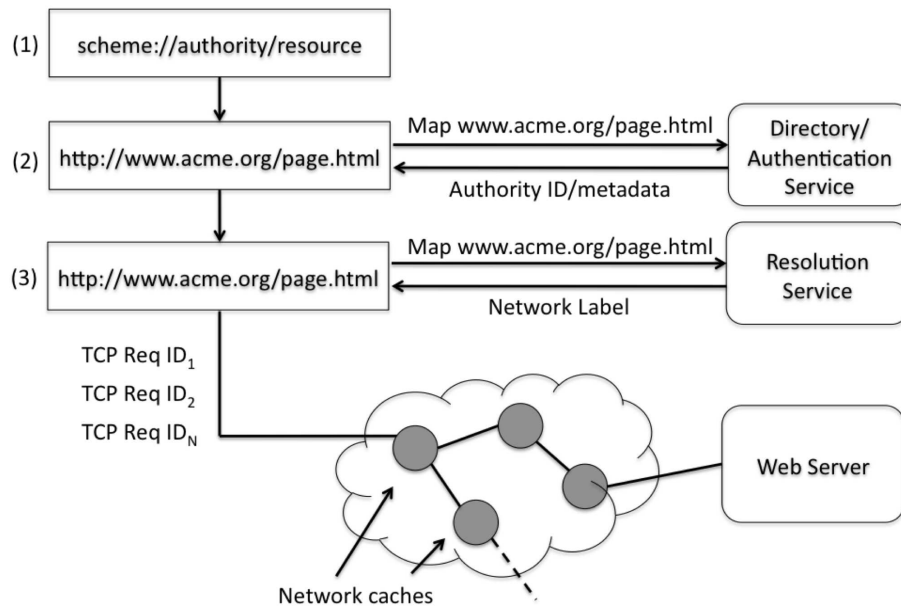


Fig. 5.2: Naming scheme and resolution to authority and metadata.

represent the trust establishment and transfer between the steps, while the solid lines represent the interactions between the content consumer and the network.

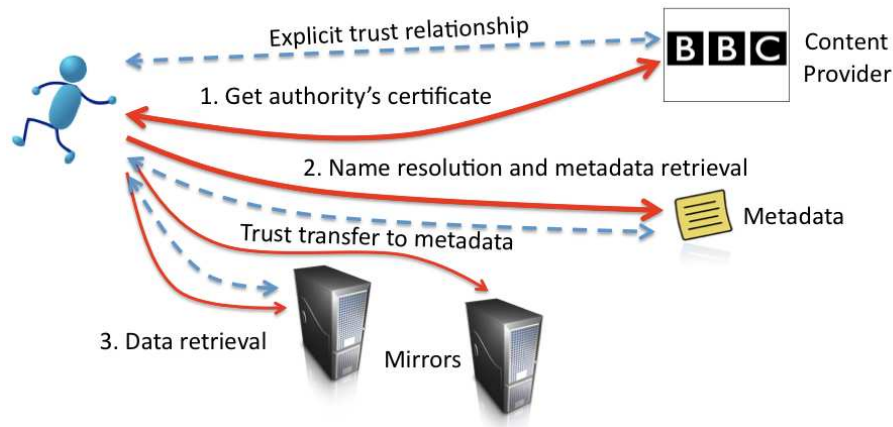


Fig. 5.3: Trust establishment and transfer from the names to the data blocks.

In order to transfer the trust and the authentication parameters from the metadata to the data blocks, we use hash trees to provide algorithmic binding between data blocks and the root hash in the metadata. Fig. 5.4 illustrates a proposal of metadata with its corresponding fields for the name-to-content resolution based on the original model proposed in Chapter 4.

As the metadata was authenticated in the previous step, the binding between authority identifier is transferred to the root hash, which will be used to authenticate the data blocks retrieved from the network. In order to provide data authentication from one cryptographic hash value (root hash) to a set of data blocks, we use hash tree techniques, such as skewed hash trees or composite hash trees

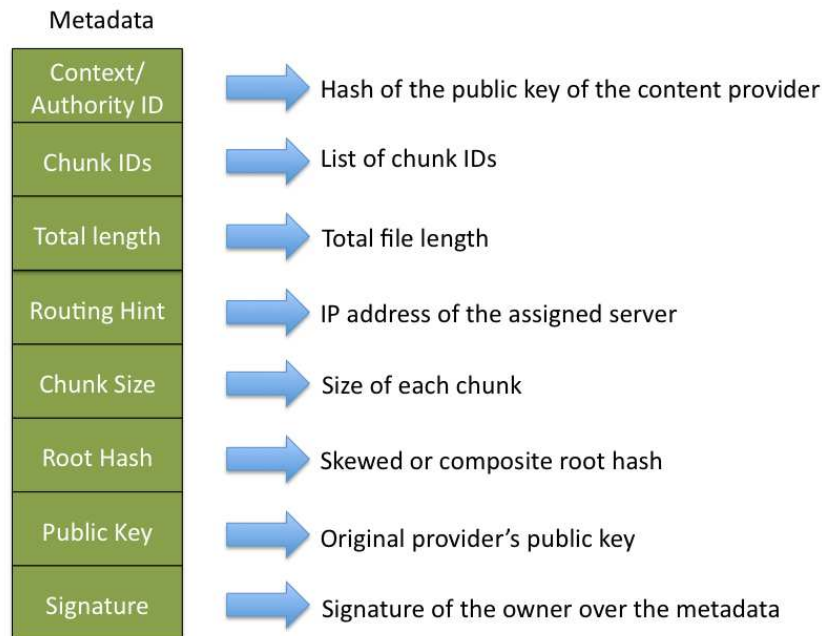


Fig. 5.4: Metadata fields used in the name resolution.

[23, 20]. Fig. 5.5 illustrates the creation of a skewed hash tree over a set of data blocks. In this scenario, each data block contains an authentication path that can be used to authenticate a data block with a root hash. If the computation of the authentication path of a data block results in the root hash that is in the metadata, then the block is authentic and was not tampered.

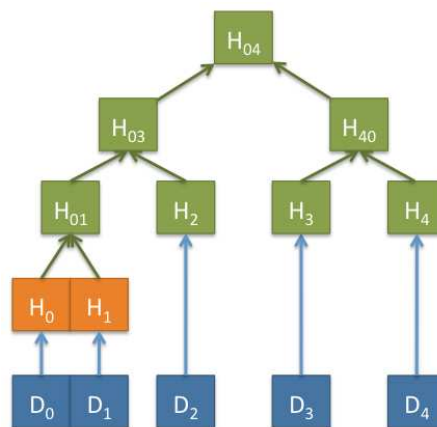


Fig. 5.5: Skewed hash tree applied over a set of data blocks.

Another possibility is to use composite hash trees to authenticate the data blocks. The benefit is the option to trade-off authentication overhead with authentication hierarchy, giving more options for the applications, for example, if the number of blocks is large, then it might be interesting to use composite hash trees.

5.1.4 Security Analysis

We analyze the proposed naming system from the external and internal security perspective. From the external point of view, we analyze the roles of the distributed directory based on the DHT and the DNS infrastructure.

The distributed directory system plays a role as digital certificate storage of authority identities, allowing clients to query for a digital certificate of a content provider. The trust placed in the directory itself is limited, since a client needs to trust that it will behave accordingly with its pre-established function, i.e., store digital certificates. One possible threat is the misbehaving of the directory system, e.g., not replying a query, resulting in lack of availability due to a malfunctioning or corrupted node. However, clients can notice such problem and try with another directory, preventing from any forgery (man-in-the-middle) attack since the trust is directly established with the content provider and not with the directory service.

Upon retrieving the content provider's certificate, clients are able to verify the digital signature in the certificate and check against the set of trusted keys in their key chain. We assume here that there is at least a small set of trusted keys in order to bootstrap the authentication procedure. Such set of trusted keys are installed by default in some Web-browsers and others can be added with the users approval.

The second resolution step involves the resolution of content IDs to network identifiers. The trust level is minimal since clients establish a direct chain of trust with the content provider. Therefore, the infrastructure is used as a simple mapping service and the possible attacks that can be performed is related to the denial of service to the clients.

From the internal point of view, we analyze the security aspects of the binding between content identifier, hash tree identifier, and the carried data. Content identifiers are based on either cryptoIDs or hash tree identifiers. In the former case (cryptoID), identifiers are generated from a strong cryptographic hash function, binding the content identifier with the data that it carries. Therefore, it is statistically impossible for an attacker to tamper a piece of data without modifying its identifier. In the latter case (hash tree identifiers), identifiers are generated through the composition of cryptographic hash functions. Similarly to the cryptoIDs, it is unfeasible for an attacker to tamper a data chunk without modifying the root hash value.

5.1.5 Deployment Considerations

The proposed naming scheme can be gradually deployed in the current naming system. From the resolution side, it required an external infrastructure acting as a placeholder for digital certificates, which can be either a DHT, e.g. Pastry [63] or an underlay forwarding/storage mechanism, e.g. PSIRP [10]. Currently, some deployed DHTs in the PlanetLab can be used for this purpose since resolvers need to place a minimal trust in the infrastructure. The benefit of using such distributed infrastructure is the resistance to some security attacks, such as denial-of-service attacks against the storage and better resistance against node failures.

For the second resolution step, it is required to introduce a new DNS type TXT record in the servers containing the content metadata. This record type is already supported by DNS, thus, servers that support the metadata scheme just need to insert a new type in the DNS to provide enough information about the data chunks.

The content servers should support both complete (or legacy request) and partial content retrieval, when a client issues a request with the content header. In the latter case, the server should have an internal mapping of the cryptoIDs to the data chunks in the complete file. For example, given a cryptoID, the server needs to be able to calculate which position of the complete file it should return, similar to the HTTP `Range` header. In this type of request, a client defines the byte range that she wants to receive from the file. Another possible approach is to save all data chunks in the server to prevent the mapping to the correct offset in the complete file.

In the client side, users need to install a plug-in in the Web-browsers to handle the new naming scheme and the metadata structure. The plug-in can be easily integrated with Web-browsers, providing authority and resource metadata authentication, metadata parsing and multiple connection management for data retrieval.

5.1.6 Summary

In this section, we have presented a secure naming system that aims at decoupling the content authentication from its location in a network. The proposed mechanism allows for content authentication using cryptographic identifiers that are independent from the routing, forwarding, and storage location. Therefore, content pieces can be authenticated with the original provider and data can be retrieved from any location, e.g., a proxy or a network cache on the path. The naming system uses a distributed directory service to store the authorities' digital certificates, allowing for the direct establishment of trust relations between clients and content providers through signed content metadata. Some benefits of the proposed naming system include the security mapping function between high-level names and cryptographic identifiers in the network level, content authentication with the provider regardless of the network location where it was retrieved, and migration of the trust from the resolution infrastructure to the provider, reducing the number of possible threats during the resolution procedure. The proposed resolution mechanism can be used as the initial name resolution to content metadata step required in the next applications scenarios, namely Peer-to-peer pollution detection and parallel authentication over HTTP.

5.2 Peer-to-peer Pollution Detection

Peer-to-peer systems (P2P) [13] were introduced as an alternative to the problematic bandwidth and processing bottlenecks in the regular client/server model. The P2P systems allowed for content storage and distribution among a set of peer nodes in the network rather than storing data in a single or a small number of servers. Therefore, a failure in one node does not compromise the system as a whole. P2P systems have scalability, fault tolerance and self organization as common characteristics and each implementation has its own topology organization, resource discovery and content distribution protocols.

The success of P2P networks is undeniable and there are many free and commercial services already deployed over their networks. Software vendors, such as Linux distributors, use P2P networks to deliver software to clients at low cost and reducing the infrastructure investment in their main servers. Despite the success, P2P systems suffer from several kinds of attacks, from resource depletion attacks [64] to service disruption [65] and data pollution [66]. Some solutions have been proposed, such as reputation systems [67], but they are limited to deal with specific problems mainly due to the open nature of the P2P networks, i.e., there is no *owner* of a public P2P network that is able to enforce a specific policy.

In this work, we analyze the pollution attacks in P2P networks and we propose a mechanism to detect partial pollution attacks using *composite hash trees*. The proposal is to create a *fingerprinting mechanism* over the data set and send it through the control plane along with the data to the peers in the data plane. Hence, receiving peers are able to detect pollution in the early stages of the download procedure, allowing them to discard and re-fetch just the part of the corrupted data rather than discarding the entire content after downloading it. This section starts with the introduction of P2P systems and their main characteristics, such as organization, content indexing and peer discovery. Then, we describe the types of pollution attacks and how they affect P2P systems. Later, we propose the *fingerprinting* mechanism based on composite hash trees [20] to enable the pollution detection in data chunks and evaluate our proposed model.

5.2.1 Background

P2P networks arose as an alternative to the problems of a regular client-server model for content distribution. In the client-server model, one or a small number of nodes concentrate the network resources requiring to implement sophisticated load balancing and fault-tolerance mechanisms to handle the increasing number of requests. As a consequence, the cost for content distribution has increased considerably and the management as well. These limitations have motivated the development of new systems that could share the processing loads and bandwidth consumption among a set of cooperative nodes. These systems are much more flexible and do not have a strict role division between clients and servers, where all nodes are considered *peers* and they can act both as clients and servers at the same time. Each peer contributes with processing power, storage capacity, software and file contents, resulting in a system with high scalability, fault tolerance and manageability. P2P architecture are composed of highly dynamic networks of peers that are organized in complex topologies and implement a special look up mechanism to find content pieces in a set of peers.

There are three main concepts in P2P networks: *resource sharing*, *decentralization* and *self organization*. The resource sharing concept implies that nodes of the system share data among them.

Thus, a resource is not strictly stored within one node, but among a set of nodes. Decentralization is a consequence of resource sharing, meaning that a piece of data is not stored in a single server, but distributed in the network. Lastly, *self organization* is reached when the P2P system is fully decentralized, thus, there is no node that coordinate other nodes and their activities in a centralized way nor have the knowledge of the whole system. Thus, nodes need to organize themselves based on the information locally available to interact with their locally reachable neighbor nodes.

P2P networks are divided in *pure* or *hybrid* model. In pure P2P model, any node can be removed from the system without any interference in the network itself. Pure P2P are inherently scalable and they are usually restricted by the number of centralized services in the system. In addition, these systems are natively fault-tolerant because there is no central point of failure and the loss of a peer can be easily compensated by another one since no node holds any special feature. On the other hand, these systems present slow information dissemination and discovery because there is no central node with the global view at the system level to coordinate these procedures. Some examples of P2P networks that fall into this category are Gnutella [68] and Freenet [69].

On the other side, the hybrid model has a central entity in the network responsible for offering services in the network. Mainly, the server maintains directories of information about peer nodes and the departure of this node affects the whole system. These systems are also differentiated by their indexing mechanism, which can be either *centralized* or *decentralized*. In the former one (centralized), a central server maintains an index of files that are currently shared among a set of peers. The benefit of such system is that peers can query the server for files, speeding up the content location and retrieval procedures. However, they present the limitation of being a single point of failure. A famous example of such system is Napster⁵. In the latter one (decentralized), a set of special nodes, known as *supernodes*, are part of a decentralized indexing system and maintain the information of active peers in the network. Thus, content queries are sent to these supernodes and they answer with a list of peers that have the queried content. An example of decentralized indexing systems is Kazaa [70].

Peer-to-peer discovery mechanisms have three generations: a first generation based on centralized structures, a second generation based on flooding mechanisms and a third generation based on distributed hash tables. In the centralized structure, peers query a centralized directory to retrieve information about content location. Peers query a directory and it will match the requests with the *best* available peer, where best can obey any criteria, such as fastest, nearest, most available, less loaded peers. Later, the peer connects directly to the node to retrieve the content. In the flooding-based model, P2P system does not maintain any central directory and each peer advertises their contents in the network and also floods the network to query for content. These systems are built in an ad hoc manner and peers have problems to accurately locate content in large networks. Finally, the last model uses a distributed hash table to provide a directory service that is distributed among a set of nodes. Each node holds a portion of the indexing table together with pointers to other nodes that have the knowledge or parts of the indexing table.

The efficiency of the P2P networks comes from the file segmentation procedure, where peers can redistribute small pieces of data as soon as they finish downloading them rather than waiting for the complete file download. As more peers have more pieces, the download procedure is fastened to all peers as result of multiple available sources for the same piece, allowing clients to open simultaneous connections to retrieve different pieces. Files in P2P networks are segmented into a set of *pieces* to

⁵<http://www.napster.com>

ease the redistribution process among peers. Each piece is also divided into several *chunks* or blocks and a chunk is the smallest transmission unit in a P2P network, as illustrated in Fig. 5.6. Each file has a *metadata* structure associated, containing the information about the file. The usual parameters within a metadata are the content size, number of pieces, piece size, owner or provider, the server where the metadata file is stored⁶ and a list of *piece IDs*. Each piece is identified by its piece ID, which is generated from the cryptographic hash over the piece, usually the SHA-1 cryptographic hash function. Each chunk is identified by the piece Id together with an *offset* or sequence value. The offset is used for ordering purposes since chunks do not have their own identifier.

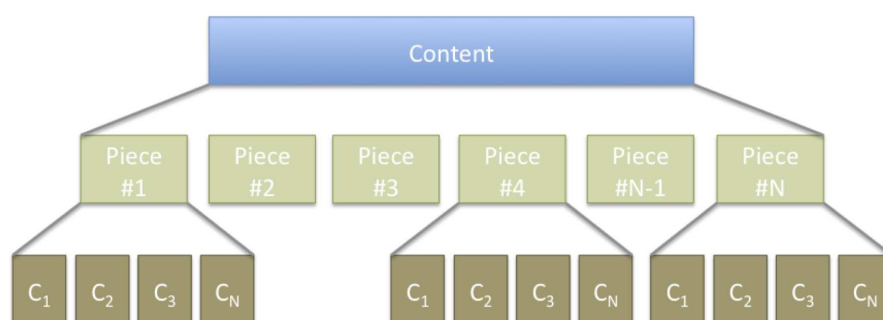


Fig. 5.6: Example of file fragmentation in P2P networks. The file is divided into a set of pieces and each piece is divided into a set data chunks.

The choice of the piece size is vital for the P2P system because it directly impacts the download time, initial content distribution and the metadata file size. The original data is sliced up into pieces and users can download smaller portions of the data and redistribute among other peers, improving the availability of that piece of content and also the overall download time for all peers. Whenever each piece is fully downloaded, it will be checked against a cryptographic hash (currently they use SHA-1 algorithm) against the metadata file. The protocol relies on *tit-for-tat* inspired protocol to spread the pieces among a set of peers. This algorithm works fine when the peer nodes have already some pieces, but there is a problem to spread the initial pieces of a content since peers do not have something to exchange. Thus, the smaller the piece size, the faster peers can start exchanging pieces.

The decision criteria to choose the piece size includes the metadata size with the pieces IDs and the total download time for a content. In the BitTorrent application, the metadata file is called *torrent*, and it should contain around 1000-1500 piece IDs in order to keep the torrent as small as 30KB. Tab. 5.1 describes the usual sizes for BitTorrent pieces⁷. Larger pieces, e.g., over 4MB can slow down the piece distribution because each piece will have 256 or more data chunks. A side effect of larger pieces is the *transmission-in-progress* state of unfinished pieces, which can't be actually shared among peers.

5.2.2 Attacks against P2P Systems

There are several attacks against P2P systems, such as free-riding [64], decreased availability [26], Sybil [71], Eclipse [65] and pollution [66] attacks. Free-riding attacks consist of a malicious node

⁶In the BitTorrent network, the centralized indexing server storing content metadata is called *tracker*.

⁷http://wiki.vuze.com/w/Torrent_Piece_Size

Tab. 5.1: Usual BitTorrent piece size

File size (MB)	350	350	350	700	1400
Piece size (KB)	64	256	512	64	1024
Number of pieces	5600	1400	700	11200	1400
Torrent size (KB)	120	30	15	240	30

consuming resource from the P2P network without contributing with any resource. As a consequence, the overall content availability decreases in the network. The Sybil attack targets the control plane of the P2P network. The idea behind this attack is that a single malicious peer presents multiple identities in the network to control part of the network. Once accomplished, the malicious node can abuse the network by corrupting a set of files and slowing down the network by rerouting all queries to a wrong direction. The Eclipse attack consists of a malicious node that takes over a certain portion of the network. Once he has achieved the goal, he can separate the network in different sub-networks and also drop queries and responses passing through the connection point. Thus, the attacker *eclipses* each sub-network from the other ones. In the pollution attack, polluters deliberately insert or replace chunks or pieces that are part of larger files, making them available with the same metadata file as the original one. This leads to receiving peer applications to discard an entire piece or content due to the corruption of a smaller fraction of the file. We will analyze this kind of attack in details and propose a mechanism to detect such attack in the next sections.

Pollution Attacks

A pollution attack in P2P networks consists of an attacker that deliberately tampers the internal content of a file, rendering it unusable and making it available in the network with the same metadata or identifier. The main problem is that polluted content can be only detected by later inspection, usually after the file is downloaded, resulting in CPU and bandwidth consumption. The pollution attack aims to undermine the credibility of the system by inserting or replacing the original searched content with another version, hoping that users just stop using the application.

Another consequence of content pollution is the increased content download times. For example, BitTorrent files are composed of pieces and each piece is composed of a set of chunks. The metainfo file contains the SHA-1 hash of the pieces, but does not contain the hash of the data chunks due to the limited size of the metainfo file. As a consequence, if there is a polluted chunk inside a large piece, there is no way to detect which one is corrupted, requiring from the application to re-download the entire piece again. For instance, in BitTorrent each piece contains 16 blocks, but the corruption of a single chunk in a piece leads to the re-download of the complete piece, wasting the other 15 blocks previously downloaded. Hence, the total download time is affected due to the repeated download of corrupted pieces, resulting in additional bandwidth consumption. The fundamental issue is the lack of binding between the original file with its data blocks, making it quite easy to corrupt any piece.

There are two main types of pollution attacks, the file pollution [72] and the index poisoning [73]. The file pollution can also be divided in two types: the complete file pollution and the partial file pollution. In the first case, the attacker replaces the list of piece IDs in the metainfo file to point to an older version of the file or to substitute to another file. In the second case, the attacker aims to insert or replace some part of the content, e.g., insert white noise in an audio file or replace the data blocks

to delay the download time. In the index poisoning attack, the malicious node advertises wrong block indexes, claiming that they own these indexes. However, these indexes do not exist, leading to fake availability. Whenever a peer node tries to search for the specified identifier, it fails to find it, resulting in the message *"More sources needed"*.

There are many proposals based on reputation systems to tackle the file pollution problem using positive feedback and blacklisting [67, 74]. They are efficient to spot polluted files in the system and blacklist them through reputation analysis, but they provide coarse granularity in their classification of the partial file pollution case.

In the total pollution attack, the polluter replaces the entire content⁸ and keeps the original metadata, and it is illustrated in Fig. 5.7. In this scenario, the total pollution attack is effective because there is no explicit trust establishment between content providers and consumers, thus, it is not possible to authenticate the metadata information retrieved from the directory server, which are almost all the cases. In this case, it is necessary an external mechanism to provide some hint about the credibility of that metadata, e.g., a reputation system.

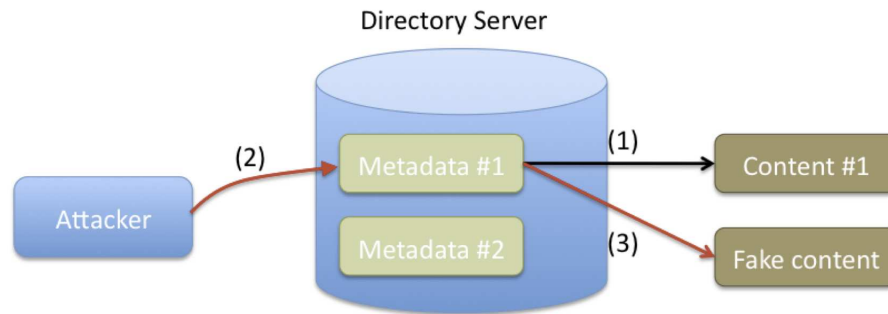


Fig. 5.7: An example of total pollution attack in P2P networks.

In the partial pollution attack, the attacker just tampers a chunk of the metadata, thus, P2P applications need to re-download the entire piece again, as illustrated in Fig. 5.8. As P2P applications just check the integrity after retrieving all chunks of the piece, there is no mechanism to detect the polluted chunk, resulting in disposal of the entire piece.

The content pollution problem is related to two main problems: the *trust* establishment between content producers and consumers and the trust transfer from the content provider down to the content fragments. In this case, the problem is that the trust is not completely transferred from the metadata down to the chunks due to a gap between content pieces and their inner chunks. In this work we will attack the partial file pollution problem by using a composite hash tree mechanism to provide the security binding between piece and chunks through cryptographic hash trees, to be presented in the next section.

5.2.3 Piece Fingerprinting Design

In order to attack partial pollution attacks, we propose a fingerprinting mechanism using composite hash trees to provide algorithmic binding between the file, its pieces and, for each piece, its

⁸A piece of content represents a file in a P2P network, and it is composed of a set of pieces and each piece is composed of a set of chunks.

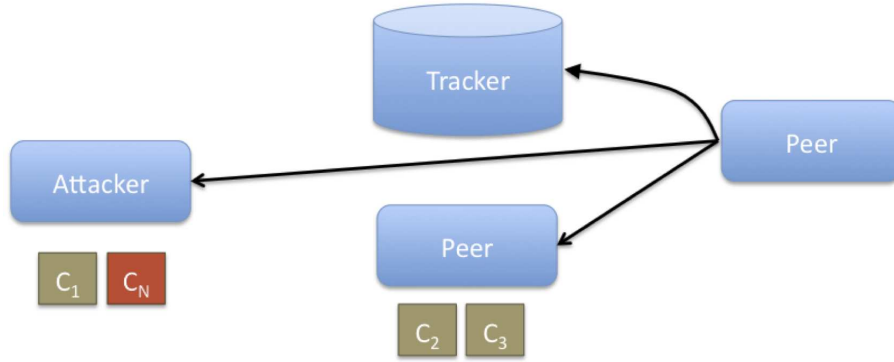


Fig. 5.8: An example of partial pollution attack in P2P networks.

internal chunks. The main idea is to create a summary tree over all the chunks and aggregate the tree information into *authentication blocks*. The root identifiers of these authentication blocks are added in the original file metadata, allowing peers to download and verify each data chunk as soon as they arrive. We assume that peers trust the metadata, i.e., the trust anchor is the metadata and it comes from a trusted peer or it was selected based on a reputation system or also it may be resolved in a name resolution mechanism previously presented in 5.1. Therefore, the root fingerprint of the composite hash tree is also trusted since it is within the metadata and it can be transferred to the data chunks.

In order to present the mechanism, we first describe the definitions used in the fingerprinting procedure. Then, we describe the fingerprinting mechanism that is used to provide the verification mechanism. Finally, we discuss the analytical evaluation of the proposed system.

Definitions

Fingerprint (FP). A *fingerprint* is the smallest authentication information required to check the integrity of a data chunk. It is generated from a cryptographic hash over a data chunk, e.g. SHA-1, and the fingerprint is strongly bound to the content. Any modification in the content carried by the data chunk will result in a different fingerprint.

Root Fingerprint (RF). The *root fingerprint* is the hash value on the top of a hash tree, representing the *fingerprint* over a data set. It is mapped to the *root hash* in the composite hash tree.

Fingerprint List (FL). The *fingerprint list* is a list containing a set of root fingerprints that are used to authenticate a set of data chunks from different hash trees. It is mapped to the *authentication data blocks* in the composite hash tree.

Chunk Verification List (CVL). The *chunk verification list* is a list of *fingerprints* needed to verify the integrity of a specific data chunk and grows linearly with the height of the tree. The CVL fingerprint in a given height h is the sibling fingerprint in the hash tree towards the root fingerprint. The main difference between a CVL and a FL is that the first one carries the verification information of one data block while the second one carries the verification data from a list of hash trees.

Initial Design

First, we provide a short review about CHT concepts and later we use the CHT technique applied to pollution detection in P2P networks. We use the composite hash tree data structure to hold all the fingerprint information required to authenticate the data and it is constructed over smaller hash trees of height h . The root fingerprints of the smaller hash trees are aggregated together in α blocks to be transferred together, minimizing the overhead of single root fingerprint transmission. Fig. 5.9(a) illustrates an example of CHT(1, 2) using HT($h = 1$) with root fingerprint aggregation of two blocks ($\alpha = 2$). These two CHT parameters, h and α , allow user to customize the data structure behavior such as the data block dependency degree for the authentication sequence or the maximum overhead desired, while providing efficient and amortized partitioned data authentication with one fingerprint comparison, previously described in Chapter 4.

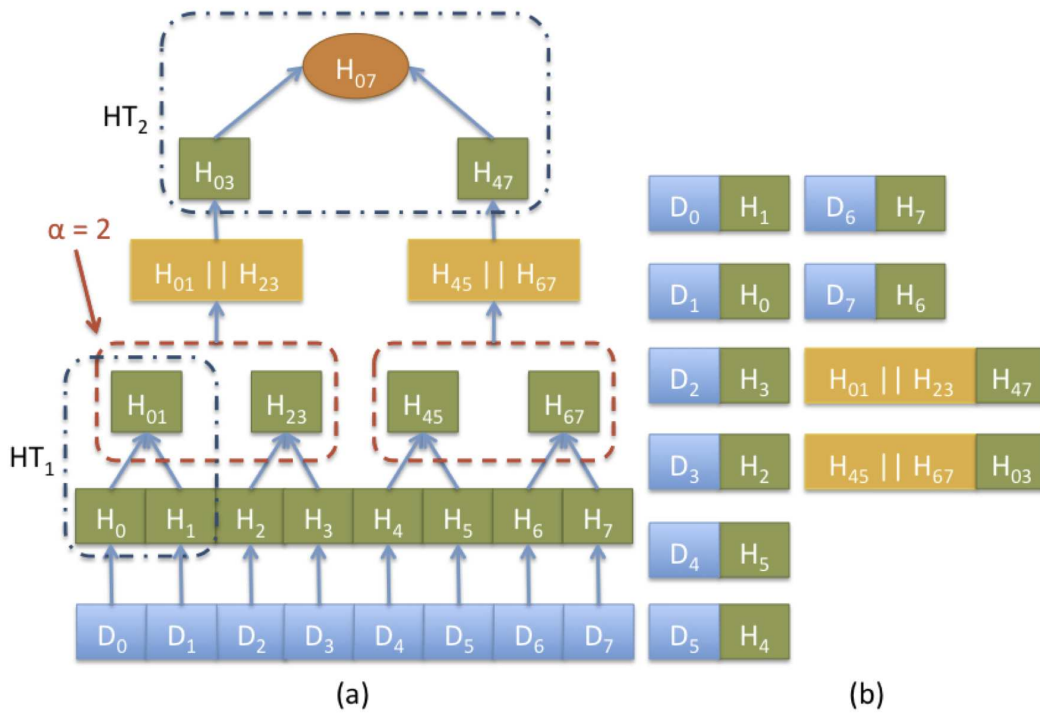


Fig. 5.9: (a) Example of a CHT($\alpha = 2, h = 1$). (b) Data chunks with their respective FVL.

The first level of the CHT(1, 2) is built over the data block fingerprints, represented in the figure by HT_1 . The root fingerprint values in the first level hash tree (HT_1) are aggregated together in blocks of two, illustrated as the fingerprint lists $H_{01} || H_{23}$ and $H_{45} || H_{67}$. Later, these lists are used as input data for the second level hash tree (HT_2) and this procedure is recursively done until the structure finishes with the root fingerprint value on the top of the tree.

Fig. 5.9(b) illustrates the FVL for each data block, containing one hash value since the hash tree used in the composition was a HT($h = 1$). In this approach, the CHT maintains just one hash value needed to provide the membership test using one fingerprint. Fig. 5.10 shows the authentication hierarchy for D_0 corresponding to the CHT(1, 2), illustrated in Fig. 5.9. The authentication procedure must start with the verification of the *fingerprint lists* $H_{01} || H_{23}$ and $H_{45} || H_{67}$ using the pre-verified

root fingerprint through some external mechanism as previously discussed, then it can proceed with the authentication of all data blocks. After the authentication of the fingerprint verification lists, they can be used to authenticate the data blocks using the same verification procedure used in the hash trees. For example, in order to authenticate D_0 , the fingerprint verification list containing H_{01} and H_{23} must be first checked with the root fingerprint.

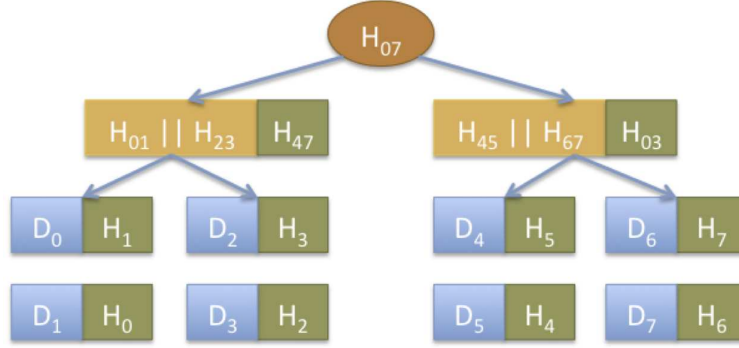


Fig. 5.10: Data chunk authentication hierarchy for CHT(1, 2).

5.2.4 Application in BitTorrent

The BitTorrent P2P application provides a metainfo file (*.torrent*) containing the URL of the tracker, the number of pieces and the pieces IDs, which are generated from the SHA-1 cryptographic hash over a piece. BitTorrent recommends the usage of *torrent* files with small size (around 30KB) to reduce the load in the servers, resulting in a number of piece IDs around 1400 in the metadata. The integration of the CHT mechanism in the BitTorrent requires minor changes in the application, and it is also possible to implement the verification mechanism as an external application.

In this section we start describing how the fingerprinting mechanism can be integrated in BitTorrent to protect against malicious changes in the data blocks. We start discussing the mechanisms for the fingerprint list distribution and *torrent* retrieval, then we present the CHT integration with the *torrent* file.

Fingerprint distribution

In order to enable BitTorrent with native block verification, users first need to create a CHT structure over the file to be shared, resulting in *Fingerprint Lists* (FL) and data blocks. The FL are distributed in the control plane as regular blocks in the network and there are two options for data block verification: *hierarchical* and *opportunistic*. In the former option (hierarchical), the FL is downloaded before fetching the data blocks, allowing the verification of data blocks as they arrive. In this option, users just store verified data blocks, but may have slower download rates until they retrieve all the FLs. In the latter option (opportunistic), clients retrieve data blocks and cache them until a FL is retrieved to start the block verification procedure. In this way, peers can start fetching data blocks prior to the reception of a FL, resulting in increased download rate but with the chance to store polluted data blocks.

The CHT-enabled BitTorrent also allows for the separation between the FL and data blocks, since there is a unique match between a *fingerprint* and the fingerprinted block. In this way, we open the possibilities for increasing the security in peer-to-peer networks, allowing peers to retrieve data blocks from any peers in the network while the verification information (FLs) comes from a trusted tracker or a security plane described in Chapter 4.

CHT-enabled *Torrent*

The first step to use CHT with BitTorrent is to retrieve the metainfo file (*torrent*) from a reliable source. In this case, we consider as reliable sources authenticated web-page of the content provider, a tracker that uses some reputation system to rank the *torrent* file or a security plane. After retrieving the *torrent*, users can query the tracker for a list of peers hosting the data blocks. Fig. 5.11 illustrates how the *torrent* file could be integrated with the CHT structure. Instead of storing the hash of an entire piece as in the regular BitTorrent, the *torrent* file stores the *Root Fingerprint* of the piece, making it possible to authenticate each chunk that is part of a piece.

On the other hand, the limitation of the number of piece IDs requires larger piece sizes, increasing the total download time [75]. For example, if a user wants to share a 350MB file, then she would create pieces of 256KB ($1400 \times 256\text{KB} = 350\text{MB}$) to be distributed among the peers, where the value 1400 represents the maximum number of piece IDs that should be present in a torrent file to keep it small, usually around 30KB. On the other hand, if she wants to distribute a file of 1.4GB, e.g., a movie, then she will need to create pieces of 1MB ($1400 \times 1\text{MB} = 1.4\text{GB}$) to maintain the reduced size of the torrent file. As peers need to download the entire piece before making it available to other peers, the size of the piece directly impacts in the download time.

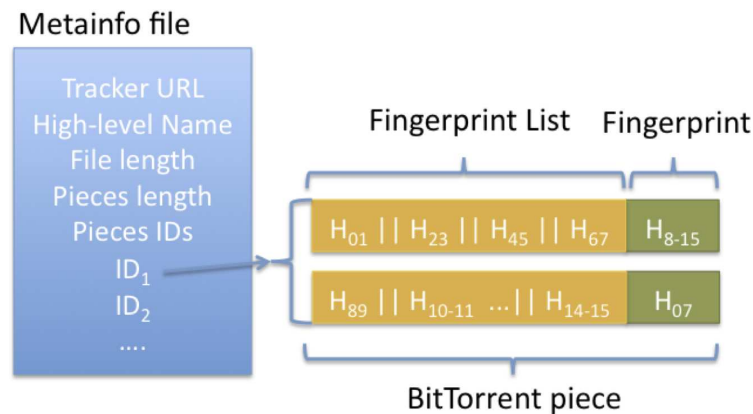


Fig. 5.11: A BitTorrent metadata file (*.torrent*) with a list of *root fingerprints*.

In the example, the ID in the *torrent* would contain the *Root Fingerprint* resulted from the hash of the FLs H₀₁..H₆₇ concatenated with H₈₋₁₅ and FLs H₈₉..H₁₄₋₁₅ concatenated with H₁₇ (similar to the hash tree procedure in Fig. 5.9). BitTorrent can verify the integrity of the FP by fingerprinting the list, concatenating it with the piece fingerprint and comparing with the piece ID in the *torrent* file. Any modifications in the list will result in a different fingerprint, making it easier for BitTorrent to spot malicious changes in the list.

Fig. 5.12 illustrates the integrity verification of data blocks using a FL. After retrieving a chunk, the application calculates the resulting fingerprint and compares with the one in the FL. As the list was verified with the *torrent* file in a previous step, the chunk is also checked using the same procedure, but with higher granularity. If the fingerprint does not match, BitTorrent can discard and retrieve just the corrupted block rather than the entire piece. Additionally, it can provide feedback for reputation systems to include possible polluters in the system into a blacklist.

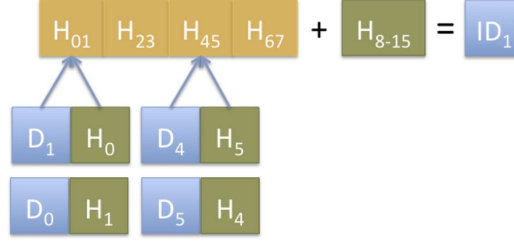


Fig. 5.12: Integrity verification of data blocks using a fingerprint list.

Fingerprinting Overhead

The fingerprinting mechanism has two overhead components associated, the chunk verification list (O_{CVL}) overhead on each chunk and fingerprint list (O_{FL}) overhead, which are the aggregated root fingerprint values of the intermediate hash trees. Thus, the total overhead is:

$$O_T = O_{CVL} + O_{FL} \quad (5.1)$$

The O_{CVL} is the sum of the product between the number of data chunk on each height by the size of the CVL, which is defined by the height of the hash tree used in the CHT. The factor $2^h \alpha$ repeats recursively i times to create the CHT over the data chunks. Therefore, the O_{CVL} formula is the product of the i recursions plus 2^h leaves over the data blocks plus the CVL length (which is the same as h) and the last hash tree created over the data blocks does not follow the pattern and adds 2^h leaves over the data blocks, thus we need to add separately in the formula to compute the overhead.

$$O_{CVL} = \sum_{i=0}^{H'} (2^h \alpha)^i \times 2^h \times (CVL \text{ length} = h) \quad (5.2)$$

where H' is the number of data blocks N minus the HT(2^h) over the data chunks that do not repeat. Therefore:

$$H' = \lceil \log_{(2^h \alpha)} (N/2^h) \rceil \quad (5.3)$$

The O_{FL} computes the sum of the product of the intermediate root fingerprint values that are aggregated into α hash values excluding the HT(2^h) over the data blocks since it starts from the first level. Hence:

$$O_{FL} = \sum_{i=1}^{H'} (2^h \alpha)^i \times (AP \text{ length} = h) \quad (5.4)$$

In order to calculate the overhead complexity with the input, first we calculate the total number of data blocks of a CHT(h, α). From 5.3, we have that:

$$N = (2^h \alpha)^{H'} \times 2^h \quad (5.5)$$

From 5.2 and substituting with 5.5, we have:

$$O_{CVL} = \sum_{i=0}^{H'} (2^h \alpha)^i \times 2^h \times h \approx (2^h \alpha)^{H'} \times 2^h \times h = N \times h \quad (5.6)$$

Therefore the O_{CVL} in the CHT is $N \times h$ and grows $O(N)$ when $N \gg h$ since h is a constant that does not change with the input size. The maximum value for h in a binary tree is $\log_2 N$, reducing the CHT to a regular Merkle Hash Tree [19] with overhead complexity of $O(N \log_2 N)$.

The fingerprint list overhead has similar proof to the previous one. Thus, substituting in 5.4, we have:

$$O_{FL} = \sum_{i=1}^{H'} (2^h \alpha)^i \times h \approx (2^h \alpha)^{H'} \times h = (N \times h)/2^h \quad (5.7)$$

Therefore, the O_{FL} in the CHT is $N \times h/2^h$ and grows $O(N)$ since h is a constant parameter that does not change with the input size. The total CHT overhead (O_T) is:

$$O_T = N \times h + (N \times h)/2^h = O(N) \quad (5.8)$$

The CHT fingerprinting mechanism can be used with many different configurations in BitTorrent. We can optimize the data structure to reduce the total fingerprint overhead by selecting hash trees with height $h = 1$ and the maximum α , which is limited by the piece length. In this case, the α index will be:

$$\alpha = \frac{\text{piece length}}{\text{hash size}} = \frac{256KB}{20B} \approx 13100 \text{ FPS} \quad (5.9)$$

In this scenario, there will be 13100 intermediate *Root Fingerprints* and the CHT height (H') will be:

$$H' = \lceil \log_{2^h \alpha} (N/2^h) \rceil = \log_{26200} 22400 \approx 0.95 = 1 \quad (5.10)$$

The total number of hierarchies is 1, meaning that if we choose the hierarchical verification procedure, there would be 1 verification procedure before verifying the data blocks themselves. The total verification overhead for the example above is:

$$O_T = N \times h + (N \times h)/2^h \times \text{hash size} = 437.25KB \quad (5.11)$$

Tab. 5.2 summarizes all the CHT overhead for different configurations. The first three options use the maximum value for α because in all cases the piece was not completely full of *Root Fingerprints*.

We can see here that using either hash trees of height 1, 2 or 3 does not make much of a difference since the file has relatively small number of blocks compared to the piece length, thus requiring one hierarchical level of authentication. By hierarchical authentication we mean that the application needs to retrieve one FL before starting downloading the pieces and blocks. In this case, the *torrent* file just need to contain the piece ID of the FL and not all the piece IDs. The fourth case does not present any

Tab. 5.2: CHT authentication overhead vs. required authentication hierarchies

CHT configuration	Overhead (KB)	Hierarchies
$(h = 1, \alpha = 13100)$	437.25	1
$(h = 2, \alpha = 13100)$	984.37	1
$(h = 3, \alpha = 13100)$	1476.56	1
$(h = 4, \alpha = 1400)$	1859.37	0

authentication hierarchy because we use larger hash trees, trading verification overhead in each data block but with lower number of *Root Fingerprints* in the *torrent* file.

Fig. 5.13 illustrates an example of a CHT($\alpha = 1400, h = 4$) where each *Root Fingerprint* can authenticate up to 16 blocks in a BitTorrent scenario. As a regular *torrent* carries about 1400 IDs, in this scenario the mechanism is able to verify a file of 350MB ($= 1400 \text{ IDs} \times 16 \text{ blocks} \times 16\text{KB block size}$) without increasing the size of the *torrent* file.

The main benefit of this approach is that the BitTorrent can start downloading immediately the blocks from the network at the cost of a verification overhead of approximately 1.82 MB (i.e., less than 1% of the file). In this case, we use a CHT(4, 1400) and each data block will carry four *fingerprints* to authenticate the data block, presenting an overhead of 80 bytes out of 16KB data block size included in the total overhead.

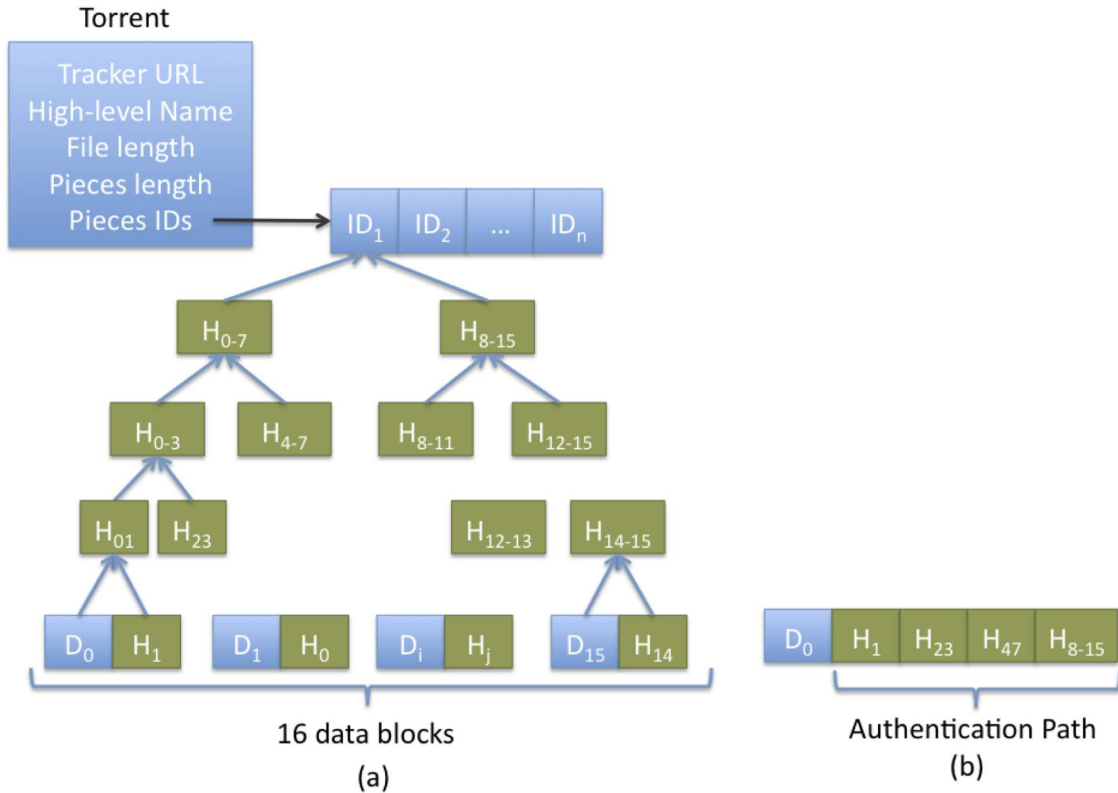


Fig. 5.13: (a) CHT ($\alpha = 1400, h = 4$). (b) Each data block has four verification fingerprints and the resulting *Root Fingerprint* of all blocks should be ID_1 to pass the integrity check.

In this scenario, BitTorrent is able to detect any corrupted data block as soon as it detects it, allowing for the application to restart the download procedure of the same block from other source.

In the second evaluation, we analyze the impact of the number of polluted blocks within a single piece. Fig. 5.14 shows a comparison between the fraction of corrupted data vs. the additional bandwidth required to download the entire file. In this evaluation, we used a 350MB file divided in pieces of 256KB and each piece was composed of 16 blocks of 16KB.

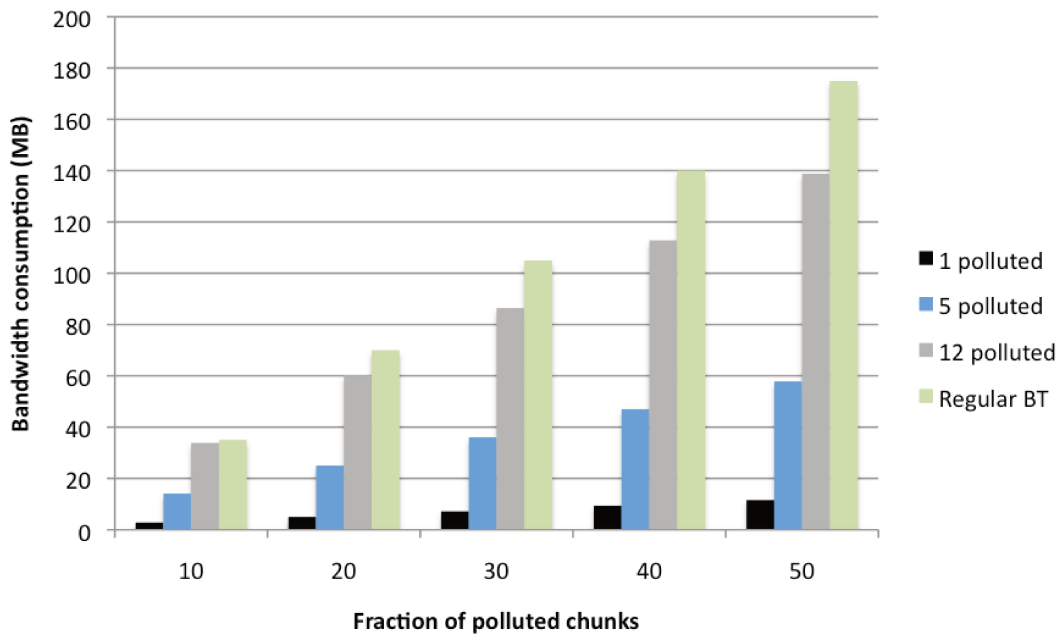


Fig. 5.14: Bandwidth consumption versus the number of polluted blocks within each piece using CHT in BitTorrent.

The figure shows that for a piece that has only one polluted block, the CHT mechanism allows for selective re-download for that piece only, resulting in reduced number of block re-downloads. Additionally to one polluted chunk in one piece scenario, we also compare with 5 and 12 polluted blocks in one piece in the regular BitTorrent. As BitTorrent re-downloads the entire piece regardless of the number of polluted blocks, it is expected that it performs worse compared to the CHT-enabled BitTorrent. In all other configurations, just the polluted block is re-downloaded, reducing the overall overhead and block retrieval. The breaking even point between using the CHT or the pure BitTorrent is approximately 12 blocks out of 16 (thus, 75% of pollution in a block). Note here that we are assuming that BitTorrent will successfully retrieve the second block from the network, which is not always true, since the second source may also send a polluted block.

We also compare the number of corrupted blocks in BitTorrent using the CHT. In this case, the overhead increases with the number of damaged blocks since the application will re-request them and perform the verification procedure again. We can see in the figure that the additional bandwidth consumption increases linearly with the number of polluted blocks per piece, while for BitTorrent it does not matter since the entire piece will be downloaded again.

Also, in the regular BitTorrent approach, the download time is not affected by the number of corrupted block in only one piece, since just one corrupted block will make BitTorrent to re-download

the entire piece again. The breaking even point between using the CHT or the pure BitTorrent is approximately 12 blocks out of 16 (thus, 75% of pollution in a block). Note here that we are assuming that BitTorrent will successfully retrieve the second block from the network without any error, which is not always true, since the second source may also send a polluted block as well.

Another benefit of the fingerprinting mechanism is the possibility to support smaller piece sizes for large files, i.e., using 256KB pieces with files larger than 1 GB⁹. In the regular BitTorrent configuration, the resulting length of the *torrent* file would be prohibitive because smaller pieces result in a higher number of IDs. The usage of CHT allows users to trade-off in-packet verification overhead (length of the *fingerprint list*) with the number of IDs that need to be stored in the *torrent* file. Based on the previous figure, users could increase the height of the tree in order to maintain the same number of IDs in the *torrent* file at the cost of verification overhead. In this scenario, we would use a CHT($h = 7, \alpha = 1400$) for a 2.8 GB file (e.g. a FreeBSD ISO), divided in 1400 pieces and 128 blocks of 16KB each ($=1400 \text{ pieces} \times 128 \text{ blocks} \times 16\text{KB (block size)} = 2.8 \text{ GB}$), resulting in an overhead of 7 fingerprints per block. Therefore, the total overhead of approximately 23.93 MB ($=1400 \text{ (maximum number of piece IDs in the metadata)} \times 128 \text{ chunks} \times 7 \text{ (height of each hash tree)} \times 20\text{B (cryptographic hash length)}$)).

Conversely, if users want to reduce the total verification overhead, they can use CHT with smaller hash trees but with hierarchical authentication of the *Fingerprint List* prior to the data block authentication. Considering the previous scenario with the same parameters, the total overhead would be approximately 1.82 MB at the cost of one level of authentication. Therefore, the CHT offers a flexible way for applications to select the structure parameters, allowing the trade-off between the authentication overhead and hierarchical authentication.

5.2.5 Summary

We have presented a pollution detection mechanism for P2P systems that can improve the detection of corrupted chunks. The partial pollution problem involves tampering one chunk of a data piece, making the whole piece invalid during the integrity check. As a result, an entire piece is re-downloaded and may be discarded all over again if the peer gets the same polluted chunk. The proposed fingerprinting mechanism allows for the detection of the corrupted chunk only, making it possible to reuse the all other correct chunks. The analytical evaluation shows that the CHT has an overhead lower than 1% of the total file size and can reduce the re-download procedure in BitTorrent in many partial pollution scenarios.

⁹BitTorrent usually does not allow for such configuration because the metadata size is larger than 30KB, resulting in slower chunk exchange in the network.

5.3 Parallel Authentication over HTTP

The introduction of P2P systems has brought many innovations in the way content could be discovered, shared and retrieved in the Internet. One of the major innovations lies on the parallel piece retrieval from multiple sources, resulting in faster download times, reduced bottlenecks in the servers and higher content availability, among other benefits. As a consequence, software vendors started to look into alternatives to reduce the load on their main content servers and one of the popular options is the *metalink* [76] proposal. Metalink is a metadata structure containing high-level information about content pieces and a list of possible sources. Therefore, applications can have more than one option to retrieve data chunks and, in case of a server failure, the application can redirect the download to another server in the list. The metalink mechanism also allows for parallel content retrieval since it also supports segmented download.

Despite the innovative approach for parallel content retrieval across heterogeneous application-level protocols, metalink relies on traditional connection-based authentication mechanisms, such as HTTPS. Hence, the security is enabled in the connection level and not in the content level, being prone to tampering attacks. In this section, we propose a lightweight mechanism for parallel authentication over HTTP using composite hash trees [23]. In the following sections, we introduce the parallel content retrieval mechanism and analyze its deployment in the Web scenario. Then, perform an analytic evaluation of the proposal and, lastly, we compare with other security proposals.

5.3.1 Background

The demand for more efficient content delivery mechanisms over the Internet has motivated the research on different approaches to improve the delivery performance. One of these systems is the content delivery networks (CDN) [12] (e.g. Akamai and Limelight) and it aims to maximize the bandwidth and content availability by placing content servers close to the possible clients. Each content server (also known as *surrogate server*) has a copy of the original content which can be delivered to requesting clients close to it. The CDN network relies on DNS redirection to map a URL to the best available surrogate servers in the network to reply to content requests. As a consequence, the CDN infrastructure is transparent to the end-users, who are actually unaware about the redirection mechanism. One downside of CDN networks is the cost associated to hire the service since CDN owners need to buy and manage the CDN infrastructure.

A second type of system is based on data retrieval from multiple sources, where the idea is to transfer the load on main servers to other sources to improve the availability and robustness in the content retrieval process. *Metalink* [76] follows this principle and borrows some ideas from P2P networks to propose a standard to implement content data retrieval from multiple sources. The Metalink mechanism is based on a metadata structure written in XML and it contains a list of addresses where clients can connect and retrieve pieces of content. In case of a failure, the application can switch from one source to another without external interference. A Metalink metadata may also contain a chunk ID list, allowing clients to fetch data chunks across different application-level protocols, for instance, HTTP, FTP and P2P protocols. In addition, there are already applications, such as MirrorBrain [77], that stores location information about nearby servers that can generate customized metalink files to clients based on their location.

Some software companies are already providing metalink files for users, such as Ubuntu and

OpenOffice, so clients have more source options to download the packages. The benefit for the vendors is the reduction on the load on their main servers since users can also use P2P protocols to retrieve data. Apple also started to implement their own protocol for parallel content download, known as *apple streaming* [78]. In this protocol, users receive a *playlist* file containing a list of URLs from where a client can download the data. Each URL points to a segment of the original data, for example, 10 seconds of a music, thus, users can fetch all segments in parallel, reducing the overall download time.

Although these two solutions, CDNs and Metalink framework, improve the performance of content download, the security mechanisms are not explicitly addressed, and they are basically inherited from the traditional security protocols. Both CDN and Metalink framework use the HTTPS as the default security mechanism to provide content authentication and integrity. For the former case (CDN), it is not actually a problem since the owner of the CDN also owns the infrastructure. Thus, the surrogate servers are considered *secure* servers and the owners are responsible for its maintenance and protection against attacks. But if there is an attack on a surrogate server and a target content is tampered, the HTTPS will not accuse any problem, since the end-points are authenticated. Unfortunately, the authenticity of the data is inherited from the authenticity of the host, which is not always true¹⁰. For the latter case (Metalink), as the content provider may not own the infrastructure that will deliver the content, e.g., a P2P network, the security issues are more critical, as malicious node can tamper the data, preventing users to correctly retrieve the content.

There is no native security mechanism to provide data authentication and integrity efficiently, leaving the client unprotected against corrupted data. One naive approach is to establish SSL/TLS tunnels with each server to authenticate the storage place. However, this approach has some drawbacks: first, it is inefficient to open multiple SSL/TLS channels, since it consumes resources on both sides, decreasing the scalability in the server; second, in this specific scenario, we are actually authenticating the storage server and not the data itself. We argue that the trust relationship is misplaced since we are placing the trust in the connection instead of the content itself.

Another approach adopted by content providers is to provide the hash digest (e.g. MD5 or SHA-1) of the entire content to guarantee the content integrity. Although this approach works well for a unicast communication scenario, where there is just one download channel, applications are only able to verify the content integrity after the complete file download, making it hard to spot corrupted data chunks in the middle of the transmission. Our goal is to increase the granularity of the verification mechanism, by allowing user applications to check smaller pieces of data as they are retrieved from the mirrors. In this case, applications can find out corrupted data faster and recover from the data corruption by selecting another server from the list and also notifying the metadata file generator about a malfunctioning/malicious node. Therefore, it is required a security mechanism to provide content authentication solely on the content information rather than source authentication in these systems.

This section has presented an amortized verification mechanism using composite hash trees [23],

¹⁰As an illustration of this scenario, consider two friends Alice and Bob. Alice trusts Bob and vice-versa and they know that they will not harm each other. Alice needs to borrow some money from Bob and Bob acknowledges that. Despite the fact that Alice knows Bob (authenticated him), there is no guarantee the bill that Bob will give to her is original or fake one (content authentication). Bob is also honest and does not want to fool Alice, but if he has received a bill that is fake and didn't realize that, he will give to Alice as a original one. Therefore, the authentication of the source does not yield to authentication of the content.

allowing applications to efficiently verify data chunks as they arrive from multiple sources. The hash tree mechanism allows for fast verification and requires just one hash computation per data segment in the best case. The proposed mechanism can be tweaked to satisfy specific application requirements, e.g., the total overhead and also the dependency between data chunks. The main difference of our approach compared to the traditional SSL/TLS-based authentication is that we enforce the content authentication and integrity based on the information that each data chunk carries instead of binding the authentication procedure to one specific source. The proposed approach has the following benefits: i) data can be more easily shared among users without requiring the verification of the serving host since the authentication information is embedded in the data; ii) fast verification, we just need one hash function per data block to check the integrity in the optimal case; iii) cheap authentication, one digital signature regardless of the number of data chunks; and iv) higher granularity to detect corrupted data chunks, making it possible to re-download it as soon as it is detected.

Metalink

The *metalink* [76] proposal aims to provide Web users with a metadata file containing information about how multiple data chunks can be retrieved from a list of sources, the geographical location of the servers and the preference level on each server. The main goal is to increase the content availability and reliability with a list of possible servers in case of a host failure. In this case, the download could be repaired and resumed using another server in the list without restarting the download from the beginning. An example of a *.metalink* file is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<metalink version="3.0"
  xmlns="http://www.metalinker.org">
  <files>
    <file name="linux-kernel">
      <verification>
        <hash type="md5">0AF...</hash>
        <hash type="sha1">38F...</hash>
      </verification>
      <resources>
        <url type="http">
          http://mirror1.com/linux-kernel</url>
        <url type="http">
          http://loc.cdn.net/linux-kernel</url>
        <url type="ftp">
          ftp://ftp.acme.com/linux-kernel</url>
        </resources>
      </file>
    </files>
  </metalink>
```

In the example above, users receive a list of possible sources and the protocols they use to deliver the data. Additionally, the metadata also contains the hash digests of the entire content, enabling the content verification just *after* the download is complete. However, our goal is to increase the

verification granularity by being able to check each data chunk just after it is retrieved instead of waiting for the complete file download.

5.3.2 Parallel Verification Proposal

We first start presenting the metrics and design goals for our parallel verification mechanism. Then, we map these requirements on the composite hash tree data structure for authentication and verification procedures. Lastly, we describe an application scenario for the composite hash tree in the parallel content retrieval context and present an analytical evaluation of the proposed verification mechanism.

Design & Rationale

In order to design a parallel verification mechanism, we considered three metrics for our model: *ordering*, *verification overhead* and *CPU processing cost*.

- **Ordering.** This metric considers the degree of dependency between the data chunks during the verification procedure. For example, hash chains [79] require *strict* ordering in the verification procedure, while per packet signature [41] or Merkle Trees [19] can provide independent packet verification (therefore, these mechanisms support *true* parallel verification).
- **Verification information overhead.** The verification information overhead, e.g., the amount of data that a packet should carry in order to provide independent verification, should be as small as possible.
- **CPU processing cost.** The verification should be fast and, preferably, at line speed.

Based on previous requirements, our goal is to have a mechanism that has none (or low) ordering requirements, low verification information overhead and low CPU processing costs. In order to achieve these requirements, we propose an authentication/verification data structure based on *composite hash trees* since it provides an efficient data verification mechanism with *low verification overhead* and *CPU processing cost* at the cost of an *initial verification ordering* requirement.

Parallel Verification Procedure

The parallel verification procedure uses the composite hash tree mechanism to provide parallel verification information retrieval together with the data blocks. The goal is to retrieve data chunks from the servers and simultaneously establish a verification relationship between the previously received data authentication blocks with the incoming ones. Fig. 5.15 shows an example of parallel data chunk retrieval and verification from multiple Web-servers.

In order to enable the parallel verification procedure in the Web, clients must first retrieve the CHT from either a trusted source or embedded in a digital certificate, illustrated in the step 5.15(a). After the verification procedure of the CHT, the client can initially open two parallel connections to retrieve the two authentication data blocks (AD) that are direct children of the CHT in the tree. After retrieving one AD, the client can verify it and open more connections to retrieve more data chunks in

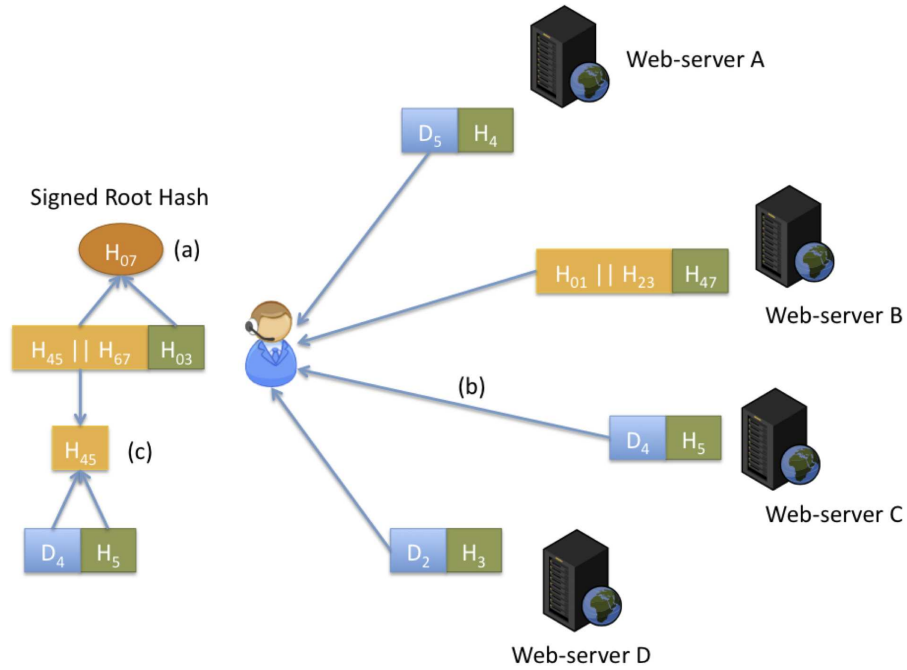


Fig. 5.15: Parallel data verification scenario. (a) First, the application retrieves the RH and verifies the digital signature. (b) The application retrieves ADs and subsequent data blocks from multiple sources. (c) Data blocks are verified using the previously received ADs.

parallel, as shown in step 5.15(b). The number of connections is limited to two in the beginning of the procedure, increasing by a factor of α connections for every AD retrieved, as illustrated in Fig.5.16. Finally, in step 5.15(c), the AD is used to verify the incoming data chunks.

Fig. 5.16 illustrates an example of data chunk verification in a client application. The figure has two columns, the first one indicates the received data chunks and the second one shows the *next chunk* window which could be downloaded next. As more ADs arrive in the client, there are more options of data chunks to be downloaded since each AD contains a list of RH that can be used to authenticate the data chunks in the hash tree leaves. Therefore, every time that an AD arrives in the left side, it is expanded and the blocks that it can verify are placed in the right column. For example, after the receiver authenticates the AD_0 containing the hash values $H_{01} || H_{23}$, the user can start downloading data blocks D_0, D_1, D_2 and D_3 in parallel and verify them as they arrive.

After the destination receives the AD with the concatenated hash values $H_{01} || H_{23}$, the receiver can retrieve and authenticate the data blocks D_0, D_1, D_2, D_3 in whichever order. The same procedure is taken when the AD with concatenated hash values $H_{45} || H_{67}$ is received in the destination, allowing the parallel retrieval and authentication of data blocks D_4, D_5, D_6 and D_7 .

5.3.3 Evaluation

In order to compare with other approaches, we perform an analytical evaluation of the CHT overhead using different configurations. As demonstrated Chapter 4, the composite hash tree has two overhead associated, the *Authentication Path* (O_{AP}) overhead and *Authentication Data Block* (O_{AD})

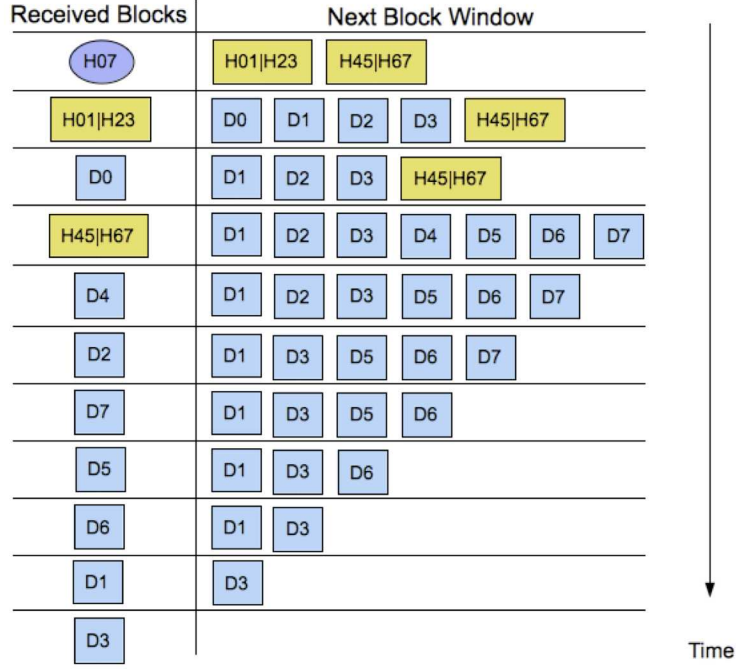


Fig. 5.16: Example of authentication hierarchy for CHT(1, 2).

overhead. The O_{AP} is the sum all AP in each intermediate hash tree, defined by the CHT height h and the O_{AD} computes the sum of the product of the intermediate RH values that are aggregated into α hash values. The total CHT overhead of a CHT (O_T) with height h and aggregation index α is:

$$O_T = N \times h + (N \times h)/2^h = O(N) \quad (5.12)$$

The CHT parameters can be tuned to fit the overhead and dependency requirements specific to applications, for instance, in delay sensitive applications, e.g., video streaming, it is interesting that we start downloading the blocks with low latency between them. As applications can open multiple channels, it can check the available bandwidth on each connection and select the one that is providing higher throughput. On the other hand, applications that are not delay sensitive, e.g., file-sharing applications, we can use CHT with higher intermediate hash trees but with lower verification overhead. In that case, smaller data blocks provide faster dissemination, and in our case, it allows us to switch faster between sources after completing a chunk download

In order to analyze the performance of CHT with different parameters, we selected a file of 1 GB which we divided in blocks of 64KB, resulting in 16384 data blocks and we chose an AD with size of 8KB. The decision to choose small data blocks is due to the possibility of switching between sources faster since we can finish one download faster in order to start with another source with higher throughput, similar to the way how P2P systems work. We first start computing the α value:

$$\alpha = \frac{\text{block size}}{\text{hash size}} = \frac{8KB}{20B} = 400 \quad (5.13)$$

Therefore, each AD will hold 400 intermediate *Root Hashes*. The hierarchy dependency will be:

$$H' = \lceil \log_{2^h \alpha} (N/2^h) \rceil = \log_{800} 8192 \approx 1.35 = 2 \quad (5.14)$$

And the total overhead will be (according to Eq. 5.12):

$$O_T = N \times h + (N \times h)/2^h \times 20(\text{hash size}) = 480KB \quad (5.15)$$

Tab. 5.3 summarizes the overhead for different h values for a file of 1 GB divided in blocks of 64KB.

Tab. 5.3: CHT overhead vs. authentication hierarchies

CHT configuration	$h = 1$	$h = 2$	$h = 3$	$h = 4$	$h = 5$
Overhead (KB)	480	800	1080	1360	1650

Hence, the total overhead for a CHT with $h = 1$ and $\alpha = 400$ is 480KB in a file of 1GB, resulting in less than 0.5% of total overhead at the cost of two verification steps before authenticating the data blocks. Another benefit from the security point of view is the fact that all blocks are algorithmically bound together, making it possible to clients to authenticate the authentication information. Compared to a regular *.torrent* used in BitTorrent, the main benefit is that we provide a mechanism to authenticate the partitioned authentication data, while the transfer of the *.torrent* file would require some other mechanism, e.g., hash chains or a single cryptographic hash over the entire metadata, to authenticate the structure containing all the piece IDs.

Fig. 5.17 summarizes the CHT overhead using different configurations of h and block sizes. Note that the overhead does not grow linearly, but logarithmically with the height of the internal hash tree (h), and the α parameter does not influence the overhead, but just the hierarchical dependency. Tab. 5.4 shows a comparison of the overhead with different file sizes and CHT configurations.

Tab. 5.4: CHT overhead (MB) vs. file size using data chunks of 64 KB.

CHT conf.	1 GB	2 GB	5 GB	10 GB	20 GB	32 GB
CHT(1, 400)	0.47	0.94	2.34	4.68	9.37	15
CHT(2, 400)	0.78	1.56	3.91	7.81	15.62	25
CHT(3, 400)	1.05	2.11	5.27	10.54	21.09	33.75
CHT(4, 400)	1.33	2.65	6.64	13.28	26.56	42.50
CHT(5, 400)	1.61	3.22	8.06	16.11	32.22	51.56
Merkle Tree	4.38	9.38	25.5	54.13	114.51	190

Fig. 5.18 shows the hierarchical dependency needed to authenticate data blocks with different h and α parameters. For this analysis, we considered a file of 1 GB divided in blocks of 64KB, resulting in 16384 data blocks. By using higher values of h , we are able to reduce the number of intermediate AD that we need to authenticate before verifying the data blocks themselves.

The graphic illustrates that for a given α value, the selection of the internal hash tree height h value does not interfere with the number of hierarchy dependencies but changes the overall overhead. For instance, if we pick $\alpha = 400$, it is equivalent to select h equal to 1, 2 or 3 since they will result in the same hierarchical dependency between blocks. However, as Fig. 5.17 shows, higher h values result in higher overhead. Therefore, the best option here is to select the smallest $h = 1$ to minimize the overhead. On the other hand, if we consider $\alpha = 50$, the value of $h = 1$, $h = 2$, $h = 3$, $h = 4$ and $h = 5$ have different hierarchical values and also overheads, being a choice of the application to select the one that best fits the application's requirements.

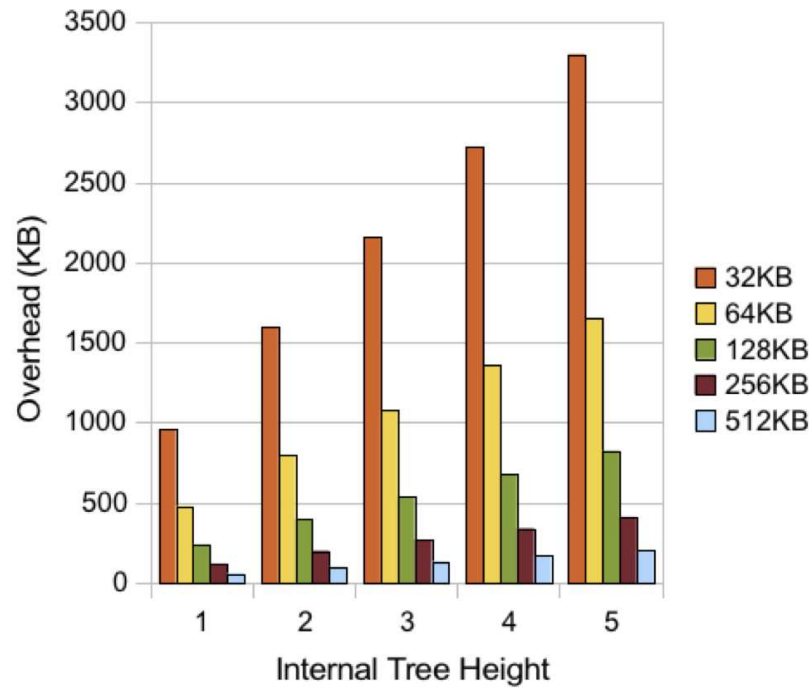


Fig. 5.17: CHT Overhead comparison using different internal hash trees for a file of 1GB divided in blocks of 32, 64, 128, 256, 512KB.

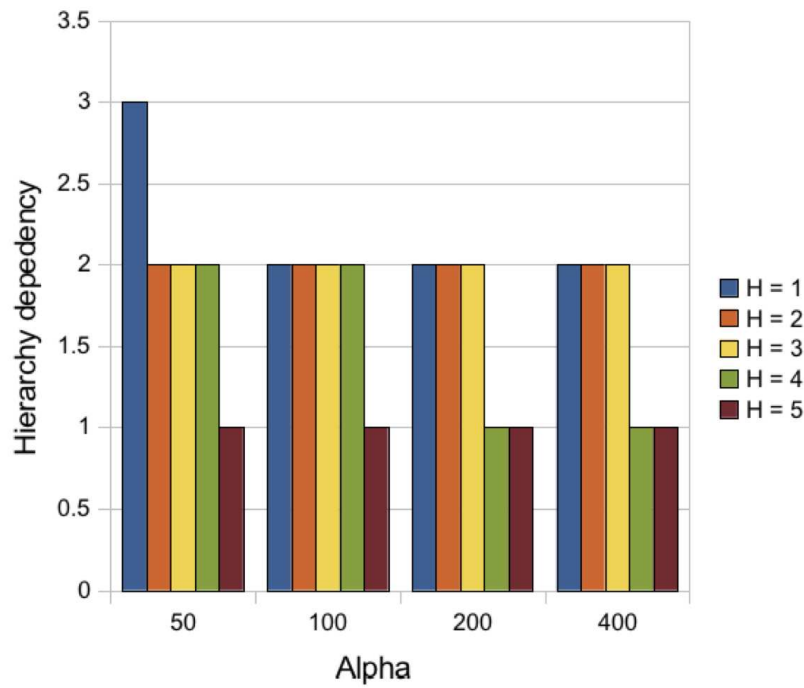


Fig. 5.18: Hierarchy dependency vs. aggregation index (α) using different internal hash tree heights.

Legacy Data Support

The proposed parallel authentication mechanism also supports legacy data from content providers, meaning that providers do not need to introduce any modifications in the files, for instance, to fragment the files into data chunks beforehand to insert the verification data (AP). As the verification data is unique and it is generated from the data segment, it is possible to detach the verification information from the data. Therefore, applications can retrieve data segments from possible sources and the AP from an authentication server or a security plane.

The content retrieval procedure from different sources starts with the metadata file retrieval from a trusted source, e.g. Metalink signed metadata or from a security plane. The metadata contains the segment sizes and the corresponding authentication ID used to authenticate the data block. Then, a client contacts a directory server to retrieve the *authentication data blocks* and the *authentication path* for each segment. Next, the client starts the verification of the AD until reaching the AP of each data block, discarding the intermediate values. In the next step, the client retrieves the metadata containing the segment sizes in order to download the segments from multiple sources using multiple protocols, e.g. HTTP and FTP. In HTTP, it is possible to use the HTTP Range Request header to request a specific segment size, and in FTP we can use the `seek` directive to request a data range. After retrieving the data segment, the application applies a cryptographic hash over the data segment and computes the intermediate *root hash* using the previously retrieved AP for the data block.

Another extension supported by the parallel retrieval is the *opportunistic* verification. The idea of the opportunistic authentication is that users start to retrieve both data and authentication data simultaneously from multiple sources instead of downloading the verification information from the authentication server. In this approach, applications do not need to wait for the AD retrieval before the data. The application just places these unverified data blocks in an outstanding table and, as soon as the verification data arrives, it checks the integrity and saves into the destination file. Fig. 5.19 illustrates the multiple source with legacy data support scenario.

5.3.4 Related Approaches

SINE [80] provides Web content integrity using a hash list scheme. The idea is to add the hash of the following block in the previous block and digitally sign the first block sent to the client, which is also known as *chain anchor*. Therefore, modifications in any of the following blocks can be spotted by computing just one hash function over the next block. The main benefits of SINE is that it requires just one digital signature to authenticate an entire piece of data regardless of the number of data blocks and use one hash function to check the integrity, resulting in both low verification header and CPU cost. The main drawback compared to CHT is the strict verification order of the pieces, therefore, not supporting parallel verification of data chunks.

Regular Merkle Trees [19] create a hash tree over a set of data blocks and each piece of data carries $\log_2 N$ hash values allowing them to authenticate data blocks with the corresponding root hash. The benefits is the independent data block verification and the low CPU processing costs. The main drawback is the verification information that each data block must carry, resulting in a total overhead of $N \times \log_2 N$, being a considerable overhead for files with large number of blocks.

Packet Level Authentication (PLA) [41] is a security model focused on per packet authentication, providing data authenticity and integrity in the network. Before a data block is sent to the destination,

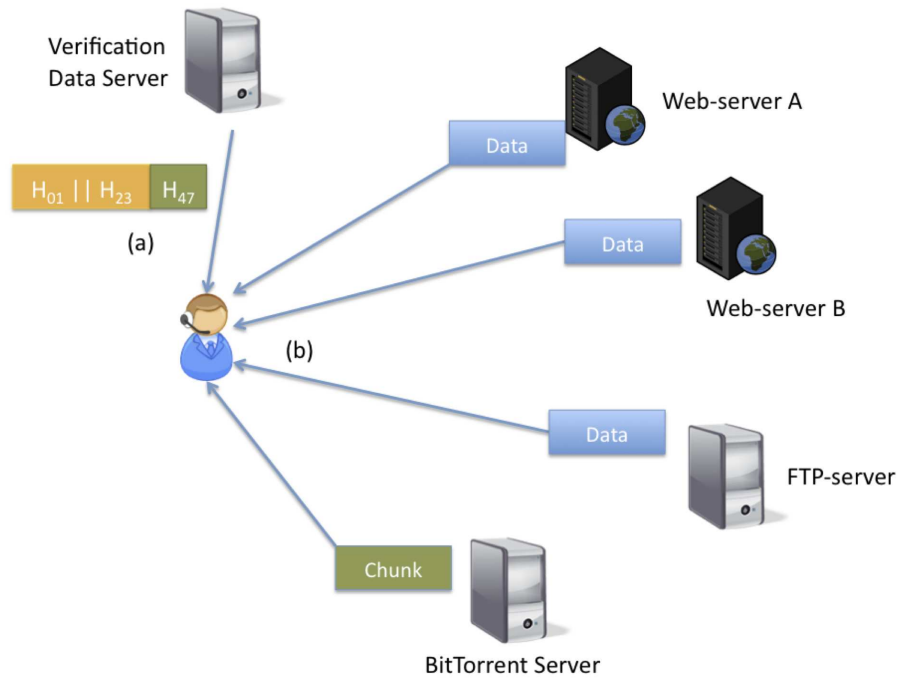


Fig. 5.19: Legacy application support using cross-application. (a) Applications can *opportunistically* retrieve data blocks from different sources. (b) Applications retrieve the *Authentication Paths* from an authentication server.

it is digitally signed by its provider, who is also endorsed by a trusted third party. The benefit is the independent block authentication with constant verification information overhead. However, the main drawback is the cost associated to the digital signature and verification, making it unfeasible to use in low processing devices. Tab. 5.5 summarizes the comparison between these mechanisms with the CHT approach. We took into account the *ordering* requirement, the *verification data overhead* and the *CPU cost* associated with the verification.

Tab. 5.5: Comparison between verification techniques

Mechanism	Block Association	Verification data	CPU cost
Hash chain	strict ordering	$O(N)$	low
Merkle Tree	independent	$O(N \times \log_2 N)$	low-medium
PLA	independent	$O(N)$	high
CHT	independent ¹¹	$O(N)$	low

The CHT mechanism inherits the low verification data overhead and CPU cost from the hash tree mechanism at the cost of an initial dependence between the first data block. After the second one, it works similarly to the regular Merkle Tree, but with linear overhead instead of $O(N \times \log_2 N)$.

5.3.5 Summary

In this work, we have presented a parallel content verification mechanism based on composite hash trees with low *ordering* requirements, verification overhead and CPU processing costs. The proposed mechanism generates the security information from the data blocks, making it agnostic from which container or mirror the data was retrieved, but just with the original source that digitally signed the metadata. The composite hash tree mechanism also allows for fine tuning of the tree characteristics, allowing the trade-off between the total verification overhead and the data block verification hierarchy. The analytical evaluation shows that the proposed mechanism has an overhead lower than 1% of the total file size and it can also be supported by legacy data without explicit modifications in the content or integrated in the metadata structure provider by content servers.

5.4 Secure Content Caching

The fast growth on the number of broadband subscribers and the increase of user generated content have put pressure on the Internet infrastructure, requiring higher bandwidth capacity and lower latency to connect content providers and end-users. Commercial incentives have increased the bandwidth availability in the *last mile*, connecting customers to the Internet, while content providers have deployed data-centers in the Internet core to provide higher bandwidth and availability in the *first mile*. However, the infrastructure connecting the residential subscribers to the content providers, also known as the *middle mile* [81], has been forgotten in the upgrade process due to the lack of incentives. Currently, the *middle mile* is the major bottleneck in the data transfer time between sites due to the lack of incentives to upgrade the intermediate infrastructure mainly composed of transit points.

In order to reduce the pressure on the infrastructure and also the inter-ISP traffic, ISPs have deployed Web caches [82], to reduce the redundant traffic going through their networks. The placement of the caches close to the consumers improves the overall user experience and also temporarily reduces the pressure on the middle mile. Content delivery networks (CDNs) [12] were proposed to leverage the caching capabilities on the network and place the content closer to the clients. Peer-to-peer caches [83] were also introduced to leverage the caching capabilities for peer-to-peer content.

On the other hand, storage prices have decreased substantially faster than bandwidth costs, indicating that it might be interesting to cache all content within a network instead of re-fetching whenever it is requested. According to [84], capacity of solid-state and magnetic storage have increased 100-fold and the costs have decreased \$50/GB and \$0.50/GB respectively, while the connectivity costs have decreased in a much slower pace [85]. In addition, recent surveys [86, 87, 88] show that a large portion of the network traffic is redundant and has the opportunity to be cached, specially peer-to-peer traffic. Despite these interesting findings, caching peer-to-peer traffic is not trivial due to the heterogeneity of the applications and protocols [84], which evolve much faster than the caching devices. Moreover, peer-to-peer protocols lack documentation, resulting in an additional barrier to deploy caches similar to Web-caches in the Internet for peer-to-peer traffic.

In this work we present an in-network caching architecture aiming at improving the overall traffic efficiency by caching authenticated content in the edge network. The general idea is to provide a forwarding fabric that forwards data requests along a set of *content routers* [24, 25] that may have stored the requested piece of data without any explicit look-up procedure. Additionally, routers along the path are able to verify whether a given piece of content is authentic, dropping the fake ones from the network using hash tree techniques. The caching mechanism uses high-level content identifiers, resulting in location-independent identifiers to represent content in the Internet. Content routers work under the application layer, thus, supporting a broader range of applications, for instance, HTTP, peer-to-peer, FTP, among others. In addition, the proposed mechanism allows for on path content lookup by forwarding requests to nearby caches, removing the lookup latency associated to Web-caches. The benefits of such approach include improved traffic efficiency by saving the amount of traffic in the network, opportunistic multi-source content retrieval by redirecting requests to nearby caches and security embedded in the content, allowing for authentication directly with the original provider through a security plane.

5.4.1 Network Caching Design

In this section, we present the in-network caching architecture, outlining the design goals for the *Content Router* (CR), and discussing the content identification, discovery, forwarding and security mechanisms.

Design Goals

The in-networking caching architecture aims at the following design goals:

- **Protocol independence.** The in-network caching mechanism must be independent of any specific protocol, for instance, peer-to-peer protocols or HTTP. This goal is mainly due to the fast evolution of peer-to-peer protocols and lack of documentation of the API, making it hard to follow and evolve together.
- **Multi-source content retrieval.** The forwarding mechanism should support multi-source content retrieval from multiple caches on the path towards the original provider.
- **Cache-based forwarding.** The delivery mechanism forwards data requests towards other in-network caches that may have the content, thus, avoiding any lookup process and incurring into a minimum latency towards the original content provider.
- **Content authenticity.** Clients should be able to verify the content integrity despite retrieving data chunks from multiple sources.
- **Authentication with original provider.** Data must be always authenticated with the original source or providers, regardless from which mirror (e.g., network cache, peer) that it was retrieved from.

Content Router

The *Content Router* (CR) is a network element that acts as a regular router and also provides content routing mechanisms. The main idea is that CRs inspect a CR header in all in-transit data and store some of them with a certain caching probability. Thus, further requests can be served by the cache data in the CR. In addition to the caching feature, CRs also store *pointers* to pieces of data that passed through it, but it decided not to cache it due to space limits. Hence, incoming data requests can be *detoured* to a neighbor CR which may have the requested piece of data, reducing the overall bandwidth consumption and latency in the network that would result by forwarding the request directly to the server. Requests can be *detoured* for a fixed amount of hops to prevent the search to go further in the network without retrieving the content. Therefore, in order to be able to cache data, CRs need to have a standardized content identification, discovery, forwarding and security mechanisms, since CRs support multi-source content retrieval. We will discuss each of these issues below.

Content Identification

In order to address resources in the Internet and cache them in the CR, we use identifiers that are simultaneously independent from the forwarding, routing, storage location and the underlying transport protocol. Thus, we use content identifiers that are solely based on the content called cryptographic identifiers (cryptoID) [7]. The benefit of using cryptoIDs are threefold: first, cryptoIDs result from a strong cryptographic hash over a data block, strongly binding the content identifier with the data that it carries; second, the cryptoID namespace is homogeneous since it results from a standard cryptographic hash function and does not need an external authority to manage the namespace; third, cryptoIDs are not bound to any specific protocol, i.e., content identification is not an internal parameter from a protocol but it exists by its own.

The basic unit of communication used in the in-network caching architecture is a data *chunk*. A chunk is a piece of data that is identified by a cryptoID with variable length. Content providers generate data chunks and use a cryptographic hash function to generate the chunks' cryptoIDs. Then, they aggregate the cryptoIDs together into meta information structure called *metadata*. The metadata also contains additional information about the content, for example, version and validity, and the chunk list is ordered to allow the correct reconstruction of the original content, as described in Chapter 4. Therefore, clients need to retrieve the content metadata prior to the data chunks download from a trusted place, e.g., a security plane described previously. For legacy applications, we use CR-proxies to perform the name to metadata resolution and the chunk retrieval (described below).

Content Discovery & Forwarding

Each CR holds an internal table called *neighborhood table* that holds information about incoming and outgoing chunks. For each incoming chunk, the CR stores the incoming interface, chunk cryptoID, outgoing interface and timestamp. Fig. 5.20 illustrates an example a neighborhood table of a CR.

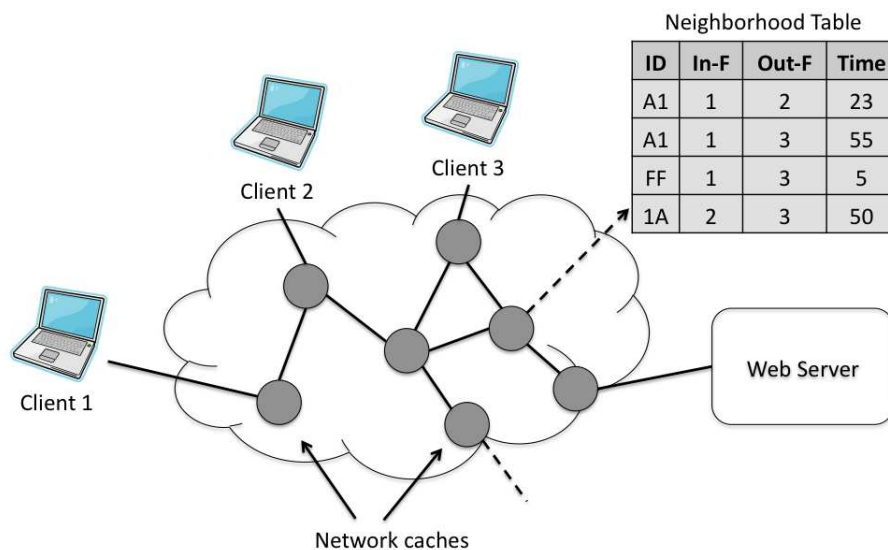


Fig. 5.20: Content routers with internal neighborhood table.

The incoming and outgoing interface information is used to forward chunk requests to the next CR which may have the requested chunk. The design decision to include both incoming and outgoing interface is to allow content lookup in different regions with pre-established precedence, for instance, towards the network edge or the core. CRs receive chunks in the incoming interface and forwards them through the outgoing interface, thus, caches can be configured to just forward requests to the outgoing interface, not the incoming interface. As a consequence, chunk requests are forwarded towards the network edge, reducing the load on the core CRs. The timestamp field contains the time when the chunk was last seen, providing information for forwarding decision and neighborhood entry eviction. In case there is more than one entry for the same cryptoID, the CR will use the most recently one as the preferred destination. Also, whenever the neighborhood table gets full, the least recently used entry will be purged from the table.

Whenever a CR receives a request for a data chunk, it checks whether its internal cache has the chunk or not. In the former case (cache has the chunk), it will respond the request on the server's behalf and return the chunk. In the latter case (cache does not have the chunk), it will look for an entry in the neighbor table to forward the request to the next content router. If there isn't an entry in the neighborhood table, it will forward the data using the underlying forwarding mechanism, e.g., IP-based routing, toward the destination. Any CR on the path is able to intercept chunk requests and will perform the same procedure as described above.

The main idea of the *neighbor zones* is to allow CR to divert regular chunk request to CR that might have the content for n hops before going directly to the server to reduce the traffic in the middle mile and the load on the server. Fig. 5.21 illustrates an example of neighborhood table and the neighbor zones around a CR.

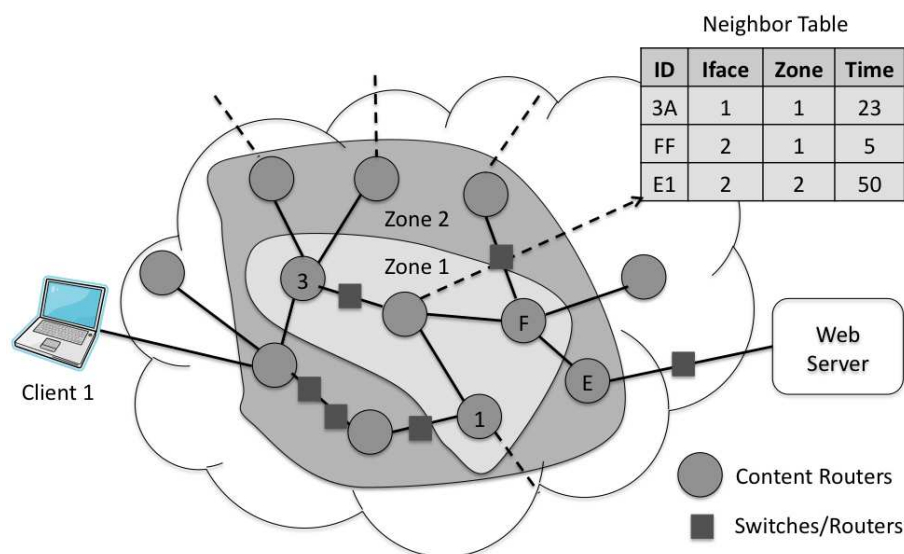


Fig. 5.21: A content router with its neighbor zones used for the chunk discovery.

CRs can be configured to introduce a limited amount of delay in the request forwarding through the neighbor zones by setting the number of neighbors that a request should visit. Thus, requests do not need to wait for look up procedures like in Web-caches but they go within the network towards other caches and, in the last case, to the server.

CR can find each other by inspecting the requests and responses in transit in the network. Each message contains a header that contains the cryptoID of the last CR, thus, the next hop CR can know its neighbors and forward requests to them. CR failures can be dealt with error messages provided by the underlying forwarding mechanism, for instance, ICMP *host unreachable messages* sent by routers. Whenever a router forwards a request towards a faulty router, the underlying IP network will return an ICMP *Host Unreachable* from the IP routers on the path. Thus, content routers on the path are able to receive the message and evict that mapping in their neighborhood table.

Neighborhood table are constantly updated with in-transit chunk to provide correct forwarding information. Whenever a request arrives on a CR, it is preferably forwarded to the outgoing interface, thus, reducing the load on the core routers. If the same request returns to the content router, it means that the content router to where the request was previously forwarded does not contain a copy of the chunk, thus, the content router removes that entry in the neighborhood table.

In order to support legacy applications, we introduce CR-proxies to bridge the legacy Internet and the CR-aware network. The CR-proxy works transparently and whenever it receives a content request, e.g., an HTTP GET, it diverts the original request to the CR-proxy and performs a name to metadata resolution. An optimization for this case is to cache the metadata in the CR-proxy, reducing the resolution step. Once the CR-proxy has the metadata, it sends multiple requests for the chunks IDs listed in the data structure, and reassembles the chunks into the original content. Then, it returns the complete content to the legacy application.

Content Security

We use the *skewed hash tree* (SHT) as the authentication data structure for the secure caching model. As described in Chapter 4, a SHT is a hash tree constructed over a set of data blocks and provide an algorithmic binding between the data blocks and the whole content. Fig. 5.22 illustrates a workflow of the secure caching mechanism using hash trees.

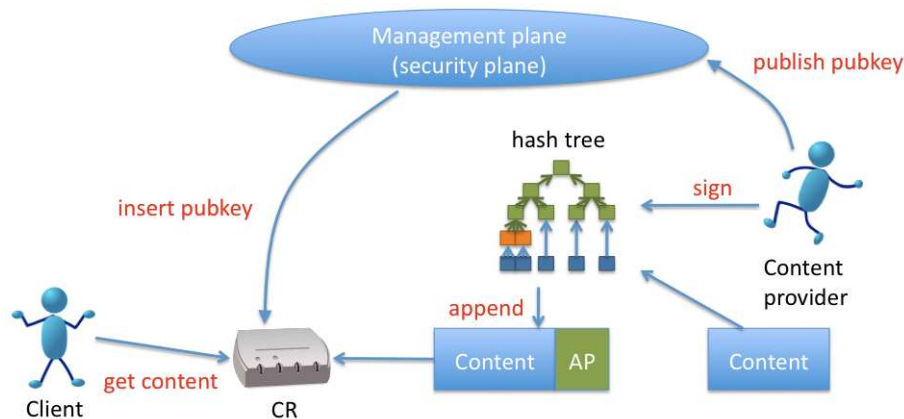


Fig. 5.22: General view of the secure caching mechanism.

In the first step, a content provider generates a SHT over a piece of content and signs the root hash of the tree over the content. Later, whenever a client request for that content, the provider sends it together with the authentication path, allowing for intermediate CRs to verify the content integrity.

CRs also need to have the provider's public key in order to verify the signature on the root hash. Therefore, we assume that CRs are managed by an ISP or a network administrator who has rights to add or remove public keys in the CR. In this scenario, administrators can obtain the public key directly from the content provider and insert into the CRs. Content providers can also publish their public keys into a security plane and administrators can manually verify their digital signature and insert them into the CRs.

5.4.2 Implementation

In this section we present the implementation of the CR mechanism. The CR is implemented as a background service running in Linux machines, composed of a kernel module and a userspace management unit, shown in Fig. 5.23.

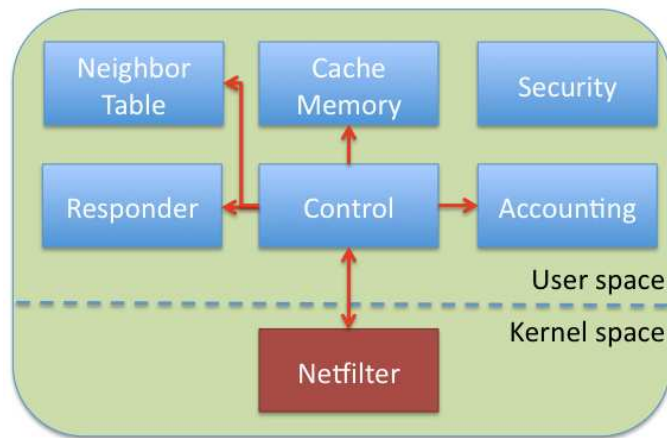


Fig. 5.23: Internal modules of a content router.

The *Netfilter* module is located in the kernel space and it is responsible for intercepting chunk request and response messages from the network, and delivering them to the *Control* module. This module uses the kernel *netlink* interface to capture packets directly from the kernel space and divert them to the user space in the *Control* module. The *Control* module handles the packet processing and forwarding, receiving data from the kernel space, caching and forwarding based on the *neighborhood table*. The *Security* module is responsible for hash tree mechanism verification using the appended authentication path. The *Cache Memory* is responsible for storing the data itself and the initial version is implemented as a hash table. The *accounting* module is responsible for collecting the statistics about the data popularity based on the requests and responses passing through the router. These statistics will be used by the *cache memory* to help the cache eviction policies. The *Neighborhood Table* contains forwarding information collected from in-transit data messages in the network together with the last-seen information. Finally, the *Responder* module is responsible for returning cached data to clients on the server's behalf.

The forwarding mechanism based on cryptoIDs between content routers use a special header containing details about the carried data. Fig. 5.24 illustrates the packet header used for the content discovery and forwarding mechanism based on cryptoIDs.

Type	Crypto ID	Cache Control	Neighbor zones	Visited neighbor	Auth. Path
------	-----------	---------------	----------------	------------------	------------

Fig. 5.24: Caching control header

The *type* field has a 8-bit field describing the type of the message, for instance, chunk request or response and signaling between CRs. The *cryptoID* is the permanent content identifier and it is generated using a cryptographic hash function, e.g., SHA-1, over the data. The *cache control* field has 8-bit length and provides signaling information for the routers, for example, whether a data chunk has already been previously cached in the network. In this case, the *cached* flag has one bit and it is stored within the *cache control* header. The *neighbor zone* field has 8-bit length and contains the number of neighbors that a message should visit before going directly to the server.

At each router, the neighbor zone field is decreased in one and, when it reaches zero, the content router forwards the data directly to the server. This field also avoids infinite loops in the network, working also as a time-to-live for a chunk message. The *visited neighbor* field has a 256-bit length *bloom filter* [89] that contains all the CR IDs visited by a chunk request to prevent loops in the network. Before forwarding a request, each CR inspects whether its cryptoID is in the *visited neighbor* field. If it is not included, then it adds cryptoID and forwards the request to the next CR, otherwise, it might be a loop or, less likely, a *false positive*. A *false positive* in an incorrect answer returned from a bloom filter when it is almost full, for example, returning that the next hop is already visited despite of not visiting it before. The false positive does not affect the CRs and the worse results is to forward the request to the original provider. The visited neighbor field also allows CRs to decide where to forward chunk requests, e.g., if a CR forwards a chunk request to an outgoing interface and the request returns without any data response, the CR will forwarding to the incoming interface to search for the content in the upper networks. In addition, the CR can purge that entry in the neighborhood table. The *Auth. Path* field contains the variable length authentication path for data verification.

The current version of the CR is implemented over UDP datagram as the forwarding mechanism, running on ports 22000 and 22001 in Linux OS machines. Clients send data requests to the servers and intermediate CRs cache these information in the popularity table, as they will be used as input parameter for caching policies. Whenever a CR intercepts a passing-by request or response, it may cache it based on the caching policies, e.g., popularity of the requests and responses. If a CR does not cache a data chunk, it adds a pointer in the *neighborhood table*, indicating possible destinations for content lookup. Whenever there is a data chunk message, CRs have a probability to cache it in their cache memory to serve for further requests.

5.4.3 Evaluation

In this section, we evaluate the CR proposal regarding the security mechanism based on the SHT.

Experimental Set-up

In order to evaluate the CR authentication mechanism and to compare with per packet signature scheme, we implemented a CR prototype in C language. We used a Mac OSX 10.6, 2.16GHz, 2 GB

RAM for the evaluation scenarios. In the first scenario, we evaluated the speed of the RSA public key signature and verification times and the SHA-1 hash function using the OpenSSL cryptographic library. The purpose of the evaluation is to establish the magnitude between a hash verification time and a digital signature and verification times. For the second, third and forth evaluations, we used the topology described in Fig. 5.25. The topology is composed of a client, a server that sends some data to the client and a CR in the border of the network where the client is located. For each test case, we collected 10 samples and considered the average value to plot the graphics.

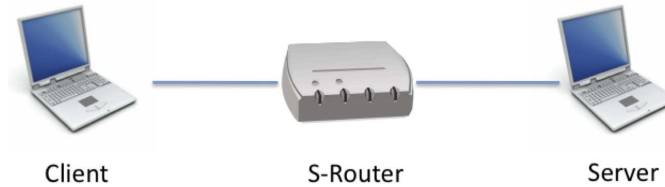


Fig. 5.25: Evaluation topology with one CR.

5.4.4 Experimental Results & Analysis

Tab. 5.6 shows the experimental evaluation of different cryptographic algorithms for signature and verification. For the SHA-1 verification speed, we considered a packet of 1024 bytes. As the experimental results show, a digital signature costs roughly 388 times slower than a hash verification and 18,51 times slower than a hash verification (SHA-1 vs. RSA 1024). The comparison is to show that if we can reduce the number of digital signatures in a large file transfer, we can considerably reduce the processing overhead resulted from the verification process. In addition, clients generating data wouldn't suffer from the delay due to the signature process.

Tab. 5.6: Signature and verification speeds with different cryptographic algorithms

Type	Signatures/s	Verification/s
SHA-1	-	222,402
SHA-256	-	96,759
RSA 1024 bits	573	12012
RSA 2048 bits	95	3601
ECC 160 bits	4830	1044
ECC 163 bits	1376	563

In the second evaluation scenario, we analyzed the root hash generation time using the SHT algorithms with the SHA-256 cryptographic hash function. We selected files ranging from 10 to 50 MB and used block sizes of 1, 2 and 4KB in the algorithm. The results are summarized in Fig. 5.26.

The figure shows that the *Root Hash* computation grows linearly with the file size and the number of data blocks. This result is predicted since the number of hash computations in the hash tree is linear to the number of data blocks. Note that the root hash has an equivalent functionality as the public key in the PKI, since it is used to verify the authenticity of a signature, but with much faster computation time.

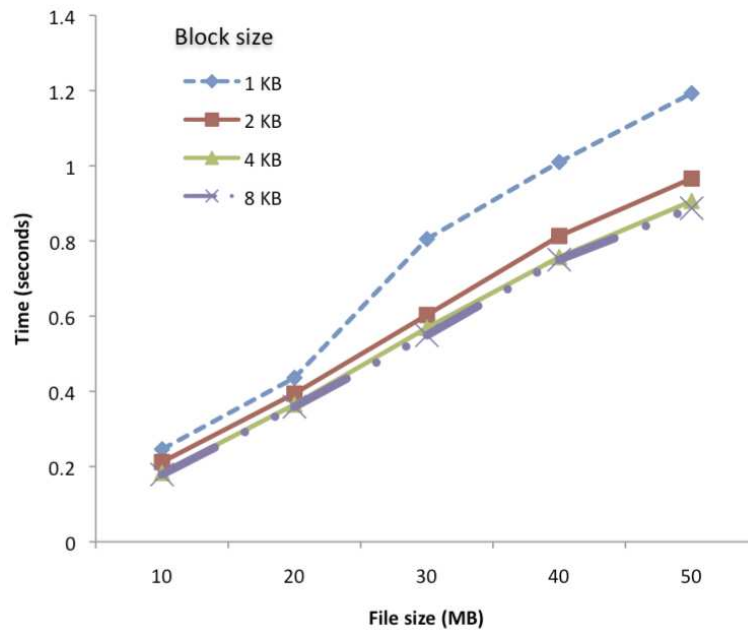


Fig. 5.26: Skewed Hash Tree Root Hash Generation Times.

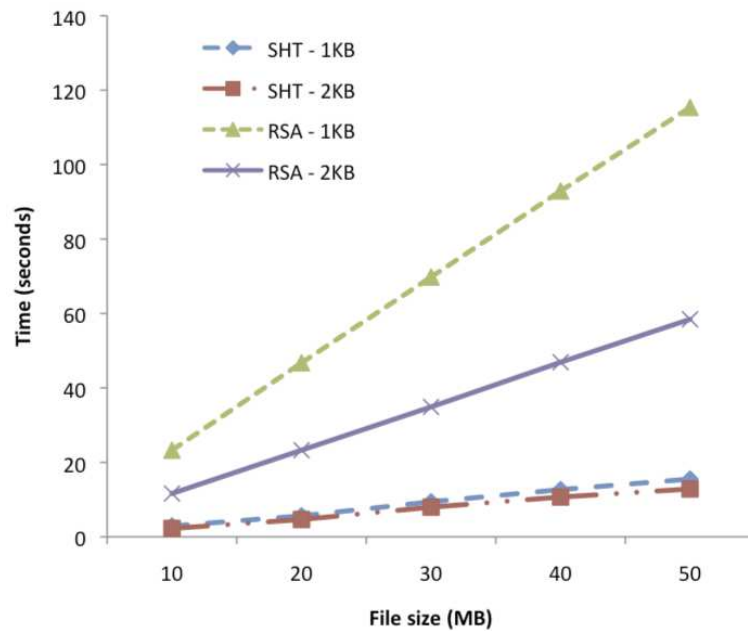


Fig. 5.27: Skewed Hash Tree AP generation times and 1024-bit RSA signature times comparison.

In the second evaluation, we compared the SHT authentication path generation time with a 1024-bit RSA signature time, shown in Fig. 5.27. We implemented two applications for this evaluation: the first one reads from an input file in blocks of 1, 2 and 4 Kbytes and apply the skewed hash tree function, and the second one reads from an input file in blocks of 1, 2 and 4 Kbytes and digitally sign each block with a 1024-bit RSA key. Both of them used the SHA-256 cryptographic hash algorithm

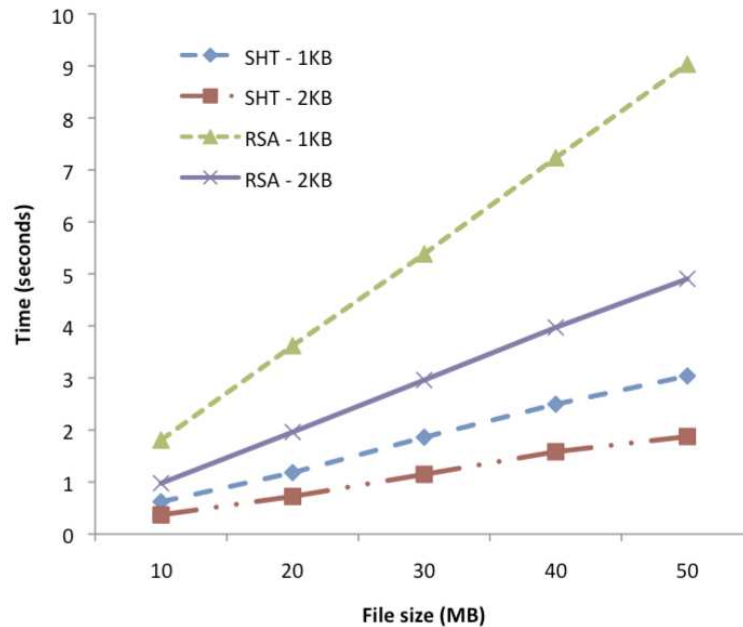


Fig. 5.28: Skewed Hash Tree AP verification times and 1024-bit RSA signature verification times comparison.

to produce the digest messages to be signed. We excluded the 1024-bit RSA key generation time from the results, since we generated it once and used the same key in the evaluation.

The results show that SHT mechanism is on average 8 times faster than the per packet signature approach. This result is expected since the digital signature computation uses large prime numbers, requiring high processing in the CPU. On the other hand, hash functions rely on bit shifting to generate the digests, resulting in lower power consumption and memory storage.

In the third evaluation, we compared the authentication path verification time in a CR with a 1024-bit RSA verification time considering different file and block sizes. We used the same applications, block size and cryptographic hash function as in the previous scenario. The experimental results from both applications are summarized in Fig. 5.28. The verification times in the SHT is on average 3 times faster than the per packet signature scheme, which are expected since the hash value computation is much faster than the public key cryptography.

5.4.5 Summary

This section has presented a secure caching mechanism based on content routers and SHTs. Content routers allow for in-transit data caching, reducing the bandwidth consumption and network latency in the content retrieval. The SHT mechanism allows for content authentication prior to the caching event, preventing from the caching of bogus or polluted data. The SHT mechanism provides a list of hash values that allows intermediate devices to verify the data independently from the source where the data came at a low cost. As proof-of-concept, we implemented a library containing the skewed hash functions to work in the CR and evaluated it in different scenarios. The experimental evaluation showed that SHT is, on average, 8 and 3 times faster in the signature and verification times

respectively, compared to per-packet signature scheme.

5.5 Summary

Chapter 5 has presented the application scenarios for the skewed hash trees and composite hash trees. The chapter started describing the name resolution mechanism to resolve content names into a secure metadata structure. The name resolution mechanism maps an authority and a content identifier into a metadata, which can be used to retrieve the network-level content chunks. Next, we present two applications of the composite hash tree in the current Internet: pollution detection in P2P networks and parallel authentication over HTTP. In the first application, a CHT is used as a piece fingerprint mechanism, generating a fingerprint for each chunk within a content piece. Hence, receiving applications are able to detect polluted blocks and replace them without discarding the correct ones. In the second application, we use the CHT mechanism to authenticate parallel HTTP connections. The main idea is to provide data authentication regardless of which connection a data block as come from. This is accomplished because CHT uses authentication information that is generated based on the content rather than on the connection. Lastly, we have presented an application of the skewed hash tree mechanism in secure content caching. The main goal is to provide data verification mechanism prior to caching, preventing from the bogus or corrupted data caching.

Capítulo 6

Conclusions

This work has presented new approaches for data authentication in information-centric networks. The proposed model aims at efficient data authentication in scenarios where the network attachment point is not previously known (spatial decoupling) and the content consumption time is not synchronized with the production (time decoupling). The initial security model aimed at providing strong cryptographic identifiers for end-hosts over the Internet and also solving the IP semantic overloading problem. These identifiers result from a strong cryptographic hash function over the public key of an end-host, strongly binding end-hosts identification and identifier. In order to generalize the idea of identification, we introduce a new layer in the network stack called identification layer. This layer is responsible for providing permanent and unique identifiers for the transport layer, leaving the network layer with the unique role of locating an end-host in the Internet. In order to provide security mechanisms, we have proposed IDSEC, an identification layer security that embeds a security protocol with denial-of-service resistance capabilities. The IDSEC handshake contains a cost function mechanism that allows for computational asymmetry between clients and servers. Hence, servers can increase the cost to communicate with them by increasing the computational costs of these challenges. As a proof of concept, IDSEC was implemented and evaluated in performance and mobility scenarios, showing that the new identification can provide native mobility support with security embedded. We have concluded that such mechanism can make end-hosts more robust against denial-of-service attacks. However, this mechanism does not protect against unwanted traffic in the network.

In order to generalize the data authentication mechanism, we have extended the security from the end-host to the content itself, mainly due to the fact that many servers just store content instead of producing it in information-centric networking. In order to tackle this limitation, we have proposed to separate the data from the security information into separate planes: a data plane and a security plane. The security plane is responsible for the security data storage and authentication. Publishers store authenticated metadata information in the plane and subscribers access the security plane to recover the content metadata. Hence, the security plane works as a trusted binding point between content providers and consumers, filling the communication gap due to the time decoupling characteristics of these networks. In order to transfer the trust from the content provider to the data blocks, we proposed two techniques to satisfy the security requirements in information-centric networks: *skewed hash tree* (SHT) and the *composite hash tree* (CHT). SHT is an extension to the Merkle Tree and allows for random file size authentication and out-of-order data verification. Additionally, SHT can be created over a set of data blocks and requires one digital signature to authenticate all data blocks. CHT is

a novel hash tree technique for data authentication with reduced authentication overhead compared to SHT. CHT uses smaller Merkle Trees to reduce the overall authentication overhead at the cost of some hierarchical authentication dependence. Once the content provider has chosen the technique, it embeds the root hash value into the metadata, which is digitally signed by the provider. Therefore, the trust is transferred from the digitally signed metadata to the data blocks.

In order to validate the security plane proposal, we implemented these two techniques in different scenarios: name resolution, pollution detection in P2P networks, parallel authentication over HTTP and secure content caching. In the first scenario (secure name resolution), we aimed at the correct name to authority and content identifiers resolution. The name follows the scheme proposed by URI and maps the authority section to the content publisher and the resource path to the metadata through the DNS or a distributed directory system. Once the client has the metadata, he is able to recover the data blocks from the data plane and authenticate them using the SHT technique. In the second scenario (pollution detection in P2P networks), we have used the CHT mechanism to detect eventual polluted blocks. For each data chunk that was generated, a providing peer also append a fingerprint to allow data authentication. Upon receiving a data packet, peers could immediately verify the integrity and reject any polluted data without discarding all pieces. The main appeal of this technique is the possibility to integrate the mechanism to the BitTorrent application, since we could add the CHT root hash in the .torrent file and the authentication path of data blocks into the BitTorrent pieces themselves. In the third scenario (parallel authentication over HTTP), we have applied CHT for parallel content authentication in the Web to provide content authentication in content delivery networks. In these networks, users can retrieve content from any mirror but they do not have any security information to validate the content. Thus, CHT fills the security gap and provides authentication tokens in the HTTP header to the clients, allowing them to authenticate the data segments regardless of the data source. Finally, in the last evaluation scenario (secure content caching), SHT has been employed to authenticate data blocks prior to the caching in network-level caches. As SHT authentication paths can be read by intermediate devices, caches can check whether a given piece of data is authentic or not prior to the caching, preventing the caching of bogus content. In the SHT evaluation, we have showed that that the signature and verification procedures are 8 and 3 times faster than regular public key cryptography routines.

6.1 Future Work

This work has focused on the application of the hash tree techniques for data authentication in many information-centric scenarios. One issue that we are going to investigate is new approaches to provide content security. An initial design is to use *convergent encryption* [90, 91] to provide confidentiality in some restricted scenarios. The convergent encryption technique uses the hash of the content as the encryption key for a given piece of data. As a consequence, the security is solely based on the content. However, the main drawback is the fact that pieces of content have permanent encryption keys, resulting in a weak security and prone to plain text attacks. We plan to investigate alternative mechanisms to provide content-based encryption, but without this limitation.

Another open issue regards the name resolution mechanism and the privacy concern. In information-centric networks, users may connect to intermediate caches to retrieve data rather than servers, which may leak information about their current interests to non-authorized third parties. Therefore, there

must be a new model to prevent the information leakage to untrusted hosts.

A third approach is to investigate new caching strategies to improve the hit ratio in CRs. As we have described in Chapter 5, CRs can use the in-network caching and reduce the bandwidth usage and the network latency. In this initial evaluation, we have just used the *least recently used* (LRU) algorithm as the replacement policy. However, there are more elaborate eviction policies that can improve the hit ratio and the network efficiency as well.

Another topic of investigation of is *spread functions*. These functions have the property to represent a set of data blocks through a mathematical function, saving the amount of identifiers that a metadata needs to carry. These functions are specially interesting in scenarios where the number of data blocks is considerable, for instance, large data blocks that are needs to be transferred in BitTorrent. In this scenario, the use of a simple mathematical function can reduce the overall size of the torrent file. Additionally, the proposed mechanism can be used as security mechanism to prevent unauthorized users to retrieve the content, where the function is the key to retrieve the block identifiers in the network.

Conclusões

Este trabalho apresentou novas abordagens para a autenticação de dados em redes orientadas à informação. O modelo proposto visa a autenticação de dados eficiente em cenários onde o ponto de ligação da rede não é previamente conhecido (desacoplamento espacial) e o consumo do conteúdo não é sincronizado com a sua produção (desacoplamento temporal). O modelo de segurança inicial busca fornecer identificadores criptograficamente seguros para os nós finais na Internet, assim como a resolução do problema da sobrecarga semântica do IP. Estes identificadores são resultado o uso de funções de *hash* criptográficos fortes sobre a chave pública de um nó final, vinculando o identificador na camada de rede com a identificação do nó. Com a finalidade de generalizar o conceito de identificação, nós introduzimos uma nova camada na pilha pilha de protocolos de rede chamada de camada de identificação. Esta camada é responsável por fornecer identificadores permanentes e únicos para a camada de transporte, liberando a camada de rede com o papel de localizar um nó na Internet. A fim de fornecer mecanismos de segurança, nós propuseram IDSEC, um mecanismo de segurança na camada de identification que incorpora um protocolo de segurança com resistência à determinados tipos de ataques de negação de serviço. O *handshake* do IDSEC contém uma função de custo que permite prover a assimetria computacional entre clientes e servidores. Como resultado, os servidores podem aumentar o custo computacional antes de estabelecer a comunicação com os clientes para repelir ataques de negação de serviço. Como prova de conceito, o IDSEC foi implementado e avaliado considerando cenários de desempenho e mobilidade, demonstrando que a nova identificação pode fornecer suporte à mobilidade nativa com a segurança incorporada ao protocolo. Concluimos que tais mecanismos pode tornar nós finais mais protegidos contra ataques de negação de serviço.

Com a finalidade de generalizar o mecanismo de autenticação de dados, nós estendemos a segurança do nó final para o conteúdo propriamente dito, principalmente devido ao fato de que muitos servidores apenas armazenam o conteúdo em vez de produzi-lo em redes orientadas à informação. A fim de resolver esta limitação, propusemos a separação dos dados das informações de segurança em planos separados: plano de dados e plano segurança. O plano de segurança é responsável pelo armazenamento de dados de segurança e autenticação. Provedores de conteúdo armazenam metadados autenticados e assinantes acessam ao plano de segurança para recuperar os metadados de conteúdo. Assim, o plano de segurança funciona como um confiável ponto de ligação entre os provedores de conteúdo e consumidores, preenchendo o vão devido ao desacoplamento temporal na comunicação nas redes orientadas à informação. A fim de transferir a confiança a partir do provedor de conteúdo para os blocos de dados, propusemos duas técnicas para satisfazer as requisitos de segurança em redes orientadas à informação: *textit skewed hash tree* (SHT) e *textit composite hash tree* (CHT). A SHT é uma extensão da *Merkle tree* e ela permite a autenticação de arquivos de tamanho aleatório e também a verificação fora de ordem dos dados. Além disso, a SHT pode ser criado sobre um conjunto de blo-

cos de dados e requer apenas uma assinatura digital para autenticar todos os blocos de dados. A CHT é uma proposta nova de árvore de *hash* para autenticação de dados com sobrecarga de autenticação reduzida, comparado com a SHT. A CHT utiliza *Merkle trees* reduzidas com o objetivo de reduzir a sobrecarga de autenticação ao custo de autenticação hierárquica. Após a escolha da técnica pelo provedor, ele pode incorporar o *root hash* na raiz dos metadados, que é assinado digitalmente pelo provedor. Consequentemente, a confiança é transferida do metadado assinado digitalmente para os blocos de dados.

A fim de validar a proposta do plano de segurança, nós implementamos essas duas técnicas em diferentes cenários: resolução de nomes seguro, detecção de poluição em redes P2P, autenticação paralela sobre HTTP e cacheamento seguro de conteúdo. No primeiro cenário (resolução de nomes seguro), buscamos na resolução correta do nome nos identificadores de autoridade e conteúdo de forma segura. O nome segue o esquema proposto pela URI e mapeia a autoridade no provedor do conteúdo e o caminho do recurso para o metadado através do DNS ou de um sistema de diretórios distribuído. Após obter o metadado, o cliente é capaz de recuperar os blocos de dados do plano de dados e autenticá-los utilizando a técnica de SHT. No segundo cenário (detecção de poluição em redes P2P), utilizamos o mecanismo de CHT para detectar eventuais blocos poluídos. Para cada bloco de dados gerado, um nó *peer* fornecedor também acrescenta uma impressão digital para permitir a autenticação de dados. Ao receber um bloco de dados, os nós pares podem verificar imediatamente a integridade dos dados e rejeitar quaisquer blocos poluídos sem descartar todas as peças. O principal atrativo desta técnica é a possibilidade integrar o mecanismo proposto ao aplicativo BitTorrent porque pode-se adicionar o *root hash* do CHT no arquivo *.torrent* e os *authentication paths* nos próprios blocos de dados. No terceiro cenário (autenticação paralela sobre o HTTP), utilizamos o CHT para autenticação paralela de conteúdo na Web para prover autenticação em redes de entrega de conteúdo. Nessas redes, os usuários podem recuperar o conteúdo de qualquer servidor, mas eles não têm qualquer informação de segurança para validar o conteúdo recuperado. Dessa forma, a CHT preenche a lacuna de segurança e fornece *tokens* de autenticação no cabeçalho HTTP para os clientes, permitindo-lhes autenticar os segmentos de dados independentemente da origem dos dados. Finalmente, no último cenário de avaliação (cacheamento de dados seguro), a SHT é utilizada para autenticar os blocos de dados antes do seu armazenamento nos cache no nível da rede. Como as *authentication paths* podem ser lidos por dispositivos intermediários, *caches* podem verificar se um bloco de dados é autêntico antes do seu cacheamento, prevenindo o cacheamento de conteúdo defeituoso. Como prova de conceito, avaliamos os tempos dos procedimentos de assinatura e verificação e os valores experimentais mostraram que as rotinas de assinatura e verificação são 8 e 3 vezes mais rápido que criptografia de chave pública.

Trabalhos Futuros

Este trabalho apresentou aplicações das técnicas de árvores de *hash* em redes orientadas à informação. Uma questão em aberto é a investigação de novas abordagens para garantir a confidencialidade dos dados. A proposta inicial é utilizar criptografia convergente [90, 91] para prover a confidencialidade dos dados em alguns cenários restritos. A técnica utiliza o *hash* do conteúdo como chave secreta para cifrar o conteúdo, provendo segurança totalmente baseada no conteúdo. A principal desvantagem desta abordagem é o fato do uso da mesma chave para cifrar os mesmos blocos de dados, abrindo vulnerabilidades de segurança. Pretendemos investigar mecanismos alternativos para

fornecer confidencialidade baseada no conteúdo, porém sem esta limitação.

Outra questão em aberto diz respeito ao mecanismo de resolução de nomes e a privacidade. Em redes orientadas à informação, os usuários podem conectar-se a caches intermediários para obter conteúdo ao invés de servidores, possibilitando o vazamento de informações sobre os interesses dos clientes para entidades não-autorizadas na rede. Dessa forma, deve haver um novo modelo de segurança e privacidade com a finalidade de se evitar a divulgação dos interesses dos usuários sem a prévia autorização.

Uma terceira abordagem é a investigação de novas estratégias de *caching* para aumentar a taxa de acertos nos CRs. Como descrito previamente, os CRs podem usar a rede de cacheamento e reduzir o uso de banda e a latência da rede. Nesta primeira avaliação, utilizamos o algoritmo *least recently used* como política de substituição das entradas no cache. No entanto, existem políticas de substituição mais elaboradas que podem aumentar a taxa de acertos das requisições, melhorando a eficiência da rede como um todo.

Um outro tópico a ser investigado é o uso de funções de espalhamento. Essas funções possuem a propriedade de representar uma list de blocks de dados através de uma função matemática, reduzindo a quantidade de identificadores necessários em um metadado. Essas funções são especialmente interessantes em cenários onde o número de blocos de dados são grandes, por exemplo, arquivos grandes que são transferidos no BitTorrent. Neste cenário, o uso de uma função matemática para representar todos os blocos de dados pode reduzir o número de identificadores listados para apenas uma função matemática. Além disso, o mecanismo proposto pode ser utilizado como um mecanismo de segurança de controle de acesso, onde apenas os portadores da função matemática podem gerar a lista de identificadores a serem recuperados da rede.

Referências Bibliográficas

- [1] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2:277–288, November 1984.
- [2] Thomas Y. C. Woo, Raghuram Bindignavle, Shaowen Su, and Simon S. Lam. Snp: an interface for secure network programming. In *Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference - Volume 1*, USTC’94, pages 4–4, Berkeley, CA, USA, 1994. USENIX Association.
- [3] *SSL and TLS: designing and building secure systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
- [4] T. Dierks and E. Rescorla. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, August 2008.
- [5] J. Saltzer. RFC 1498: On the Naming and Binding of Network Destinations, August 1993.
- [6] David Clark, Robert Braden, Aaron Falk, and Venkata Pingali. Fara: reorganizing the addressing architecture. In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, FDNA ’03, pages 313–321, New York, NY, USA, 2003. ACM.
- [7] Robert Moskowitz, Pekka Nikander, Petri Jokela, and Thomas Henderson. RFC 5201: Host Identity Protocol, April 2008.
- [8] T.R. Henderson. Host mobility for ip networks: a comparison. *Network, IEEE*, 17(6):18 – 26, 2003.
- [9] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: defining tomorrow’s internet. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM ’02, pages 347–356, New York, NY, USA, 2002. ACM.
- [10] Publish/Subscribe Internet Routing Paradigm. Conceptual architecture of psirp including sub-component descriptions. Deliverable d2.2, PSIRP project. , August 2008.
- [11] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM ’07, pages 181–192, New York, NY, USA, 2007. ACM.

- [12] George Pallis and Athena Vakali. Insight and perspectives for content delivery networks. *Commun. ACM*, 49:101–106, January 2006.
- [13] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36:335–371, December 2004.
- [14] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005.
- [15] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35:114–131, June 2003.
- [16] Walter Wong, Maurício Ferreira Magalhães, and Fábio Verdi. IDSec: An Identification Layer Security Model. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 1093–1100, Washington, DC, USA, 2010. IEEE Computer Society.
- [17] Walter Wong, Marcus Giraldi, Maurício Ferreira Magalhães, and Fábio Verdi. An Identifier-Based Architecture for Native Vertical Handover Support. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 252–259, Washington, DC, USA, 2010. IEEE Computer Society.
- [18] Walter Wong, Fábio Verdi, and Maurício F. Magalhães. A security plane for publish/subscribe based content oriented networks. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 45:1–45:2, New York, NY, USA, 2008. ACM.
- [19] Ralph C. Merkle. A certified digital signature. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 218–238, New York, NY, USA, 1989. Springer-Verlag New York, Inc.
- [20] Walter Wong, Maurício Ferreira Magalhães, and Jussi Kangasharju. Piece Fingerprinting: Binding Content and Data Blocks Together in Peer-to-peer Networks. *IEEE Global Communications Conference (Globecom'10)*, Miami, Florida, USA, December 6-10 2010.
- [21] Walter Wong and Pekka Nikander. Secure Naming in Information-centric Networks. *3rd ACM Re-Architecting the Internet (ReArch'10) Workshop, co-allocated with 6th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT'10)*, 2010.
- [22] Walter Wong and Pekka Nikander. Towards Secure Information-centric Naming. *Securing and Trusting Internet Names Workshop (SATIN'11)*, April 4-5 2011.
- [23] Walter Wong, Maurício Ferreira Magalhães, and Jussi Kangasharju. Towards Verifiable Parallel Content Retrieval. *6th Workshop on Secure Network Protocols (NPSec'10)*, Kyoto, Japan, October 2010.
- [24] Walter Wong, Marcus Giraldi, Maurício Ferreira Magalhães, and Jussi Kangasharju. Content Routers: Fetching Data on Network Path. *IEEE International Conference on Communications (ICC'11)*, Kyoto, Japan., June 5-9 2011.

- [25] Somaya Arianfar, Jorg Ott, Lars Eggert, Pekka Nikander, and Walter Wong. A Transport Protocol for Content-centric Networks. Extended Abstract. *18th International Conference on Network Protocols (ICNP'10)*, Kyoto, Japan, October 2010.
- [26] Nicolas Christin, Andreas S. Weigend, and John Chuang. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *EC '05: Proceedings of the 6th ACM conference on Electronic commerce*, pages 68–77, New York, NY, USA, 2005. ACM.
- [27] L. Lamport. Constructing digital signatures from a one-way function. Technical report, Oct 1979.
- [28] N. Ferguson; B. Schneier. Practical cryptography. Wiley, 2003.
- [29] Roberto J. Bayardo and Jeffrey Sorensen. Merkle tree authentication of http responses. In *Special interest tracks and posters of the 14th international conference on World Wide Web, WWW '05*, pages 1182–1183, New York, NY, USA, 2005. ACM.
- [30] Tobias Heer, Stefan Götz, Oscar Garcia Morchon, and Klaus Wehrle. Alpha: an adaptive and lightweight protocol for hop-by-hop authentication. In *Proceedings of the 2008 ACM CoNEXT Conference, CoNEXT '08*, pages 23:1–23:12, New York, NY, USA, 2008. ACM.
- [31] Ahsan Habib, Dongyan Xu, Mikhail Atallah, Bharat Bhargava, and John Chuang. A tree-based forward digest protocol to verify data integrity in distributed media streaming. *IEEE Trans. on Knowl. and Data Eng.*, 17:1010–1014, July 2005.
- [32] Roberto Tamassia and Nikos Triandopoulos. Efficient content authentication in peer-to-peer networks. In *Proceedings of the 5th international conference on Applied Cryptography and Network Security, ACNS '07*, pages 354–372, Berlin, Heidelberg, 2007. Springer-Verlag.
- [33] Markus Jakobsson, Tom Leighton, Silvio Micali, and Michael Szydlo. Fractal merkle tree representation and traversal. In *Proceedings of the 2003 RSA conference on The cryptographers' track, CT-RSA'03*, pages 314–326, Berlin, Heidelberg, 2003. Springer-Verlag.
- [34] Piotr Berman, Marek Karpinski, and Yakov Nekrich. Optimal trade-off for merkle tree traversal. *Theor. Comput. Sci.*, 372:26–36, March 2007.
- [35] M. Szydlo. Merkle tree traversal in log space and time. In *Proc. Advances in Cryptology (Eurocrypt'04)*, 3027(541-554), 2004.
- [36] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. *RFC 4346*, April 2006.
- [37] S. Kent and K. Seo. Security Architecture for the Internet Protocol. *RFC 4301*, December 2005.
- [38] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). *RFC3022*, January 2001.
- [39] L. Phifer. The Trouble with NAT. *The Internet Protocol Journal*, 3(4):2–13, December 2000.

- [40] H. Krawczyk, M. Bellare and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. *RFC 2104*, February 1997.
- [41] Catharina Candolin, Janne Lundberg, and Hannu Kari. Packet level authentication in military networks. In *Proceedings of the 6th Australian Information Warfare & IT Security Conference*, November 2005.
- [42] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. *RFC 4423*, May 2006.
- [43] P. Jokela, R. Moskowitz and P. Nikander. Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP). *RFC 5202*, April 2008.
- [44] P. Nikander, T. Henderson, C. Vogt and J. Arkko. End-Host Mobility and Multihoming with the Host Identity Protocol. *RFC 5206*, April 2008.
- [45] P. Nikander and J. Laganier. Host Identity Protocol (HIP) Domain Name System (DNS) Extension. *RFC 5205*, April 2008.
- [46] Ross Anderson, Francesco Bergadano, Bruno Crispo, Jong-Hyeon Lee, Charalampos Maniavas, and Roger Needham. A new family of authentication protocols. *SIGOPS Oper. Syst. Rev.*, 32:9–20, October 1998.
- [47] Eric Renault, Ahmad Ahmad, and Mohamed Abid. Toward a security model for the future network of information. *Proceedings of the 4th International Conference on Ubiquitous Information Technologies & Applications*, pages 1–6, December 2009.
- [48] C. Dannewitz, J. Golic, B. Ohlman, B. Ahlgren. Secure Naming for a Network of Information. *INFOCOM IEEE Conference on Computer Communications Workshops*, pages 1–6, March 2010.
- [49] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.
- [50] Van Jacobson, Diana K. Smetters, Nicholas H. Briggs, Michael F. Plass, Paul Stewart, James D. Thornton, and Rebecca L. Braynard. Voccn: voice-over content-centric networks. In *Proceedings of the 2009 workshop on Re-architecting the internet*, ReArch '09, pages 1–6, New York, NY, USA, 2009. ACM.
- [51] R. Jain. Internet 3.0: Ten Problems with Current Internet Architecture and Solutions for the Next Generation. *Military Communications Conference, 2006. MILCOM 2006, Washington, DC.*, pages 1–9, October 2006.
- [52] Jari Arkko and Pekka Nikander. Weak authentication: How to authenticate unknown principals without trusted parties. In Bruce Christianson, Bruno Crispo, James Malcolm, and Michael Roe, editors, *Security Protocols*, volume 2845 of *Lecture Notes in Computer Science*, pages 57–66. Springer Berlin / Heidelberg, 2004.

- [53] J. Leiwo, T. Aura and P. Nikander. Towards Network Denial of Service Resistant Protocols. *In Proceedings of the 15th International Information Security Conference (IFIP/SEC 2000), Beijing, China*, pages 301–310, August 2000.
- [54] A. Back. Hashcash - A Denial of Service Counter-Measure. August 2002.
- [55] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram and D. Zamboni. Analysis of a Denial of Service Attack on TCP. *In In Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223. IEEE Computer Society Press, 1997.
- [56] Walter Wong, Rafael Pasquini, Rodolfo Villaça, Luciano de Paula, Fabio Verdi, and Maurício Magalhães. An Architecture for Mobility Support in a Next Generation Internet. *22nd IEEE International Conference on Advanced Information Networking and Applications (AINA'08), Ginowan, Okinawa, Japan.*, March 2008.
- [57] Walter Wong, Fabio Verdi, and Maurício Magalhães. A Next Generation Internet Architecture for Mobility and Multi-homing Support. *3rd International Conference on emerging Networking EXperiments and Technologies (CoNEXT'07), New York, NY, USA*, December 10-13 2007.
- [58] D. Eastlake. Domain Name System Security Extensions. *RFC 2535*, March 1999.
- [59] A. Botta, A. Dainotti and A. Pescapé. Multi-protocol and Multi-platform Traffic Generation and Measurement. *INFOCOM*, May 2007.
- [60] Augusto Morales, Oscar Novo, Walter Wong, and Tomas Valladares. Publish/Subscribe Communication Mechanisms over PSIRP. *7th IEEE International Conference on Next Generation Web Services Practices (NWeSP'11), Salamanca, Spain*, October, 19-21 2011.
- [61] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. Increased dns forgery resistance through 0x20-bit encoding: security via leet queries. *In CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 211–222. ACM, 2008.
- [62] T. Berners-Lee. RFC1630: Universal Resource Identifiers in WWW, June 1994.
- [63] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.
- [64] Arnaud Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. *In IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 203–216, 2006.
- [65] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstron. Defending against eclipse attacks on overlay networks. *In Proceedings of the 11th workshop on ACM SIGOPS European workshop*, EW 11, New York, NY, USA, 2004. ACM.

- [66] Cristiano Costa, Vanessa Soares, Jussara Almeida, and Virgilio Almeida. Fighting pollution dissemination in peer-to-peer networks. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1586–1590, New York, NY, USA, 2007. ACM.
- [67] Jian Liang, Naoum Naoumov, and Keith W. Ross. Efficient blacklisting and pollution-level estimation in p2p file-sharing systems. In *In AINTEC*, pages 1–21, 2005.
- [68] . Gnutella development forum. Available: http://groups.yahoo.com/group/the_gdf/files/.
- [69] Ian Clarke, Oskar S. On Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *In Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2000.
- [70] . Kazaa media desktop. J. Available: <http://www.kazaa.com/>.
- [71] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, 2002. Springer-Verlag.
- [72] Jian Liang and Rakesh Kumar. Pollution in peer-to-peer file sharing systems. *INFOCOM 2005*, pages 1174–1185, 2005.
- [73] J. Liang, N. Naoumov, and K. W. Ross. The index poisoning attack in p2p file sharing systems. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, 2006.
- [74] Kevin Walsh and Emin Gün Sirer. Fighting peer-to-peer spam and decoys with object reputation. In *P2PECON '05: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 138–143, New York, NY, USA, 2005. ACM.
- [75] Gang Wu and Tzi cker Chiueh. How efficient is bittorrent? volume 6071, page 607100. SPIE, 2006.
- [76] Metalink - Bridging the Gap. <http://www.metalinker.org/>.
- [77] MirrorBrain - A Download Redirector and Metalink Generator. <http://mirrorbrain.org/>.
- [78] R. Pantos. HTTP live streaming. Internet Draft draft-pantos-http-live-streaming (Work in Progress), 2010.
- [79] Yih-Chun Hu, M. Jakobsson and A. Perrig. Efficient Constructions for One-Way Hash Chains. *Applied Cryptography and Network Security*, pages 423–441, May 2005.
- [80] C. Gaspard, S. Goldberg, W. Itani, E. Bertino and C. Nita-Rotaru. SINE: Cache-Friendly Integrity for the Web. *5th Network Protocol Security Workshop (NPsec'09)*, 2009.
- [81] Tom Leighton. Improving performance on the internet. *Commun. ACM*, 52(2):44–51, 2009.
- [82] Pablo Rodriguez, Christian Spanner, and Ernst W. Biersack. Analysis of web caching architectures: Hierarchical and distributed caching. *IEEE/ACM Transactions on Networking*, 9:404–418, 2001.

- [83] Mohamed Hefeeda, Cheng-Hsin Hsu, and Kianoosh Mokhtarian. pcache: A proxy cache for peer-to-peer traffic. In *SIGCOMM 2008*, page 539, August 2008.
- [84] Decoupled Application Data Enroute (DECADE). URL <http://datatracker.ietf.org/wg/decade/>.
- [85] Anirudh Badam, KyoungSoo Park, Vivek S. Pai, and Larry L. Peterson. Hashcache: cache storage for the next billion. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 123–136, Berkeley, CA, USA, 2009. USENIX Association.
- [86] Jeffrey Eрман, Alexandre Gerber, Mohammad T. Hajiaghayi, Dan Pei, and Oliver Spatscheck. Network-aware forward caching. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 291–300, New York, NY, USA, 2009. ACM.
- [87] Ager, B.; Schneider, F.; Juhoon Kim; Feldmann, A. Revisiting cacheability in times of user generated content. *IEEE INFOCOM Conference on Computer Communications Workshops*, pages 1–6, March 2010.
- [88] Ashok Anand, Chitra Muthukrishnan, Aditya Akella, and Ramachandran Ramjee. Redundancy in network traffic: findings and implications. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 37–48, New York, NY, USA, 2009. ACM.
- [89] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
- [90] Alan Mislove, Ansley Post, Charles Reis, Paul Willmann, Peter Druschel, Dan S. Wallach, Xavier Bonnaire, Pierre Sens, Jean-Michel Busca, and Luciana Arantes-Bezerra. Post: a secure, resilient, cooperative messaging system. In *Proceedings of the 9th conference on Hot Topics in Operating Systems - Volume 9*, pages 11–11, Berkeley, CA, USA, 2003. USENIX Association.
- [91] John R. Douceur, Atul Adya, William J. Bolosky, Dan Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, pages 617–, Washington, DC, USA, 2002. IEEE Computer Society.