2011

CONSTRUCTION OF EFFICIENT AUTHENTICATION SCHEMES USING TRAPDOOR HASH FUNCTIONS

Santosh Chandrasekhar
University of Kentucky, san@netlab.uky.edu

ABSTRACT OF DISSERTATION

Santosh Chandrasekhar

The Graduate School
University of Kentucky
2011

CONSTRUCTION OF EFFICIENT AUTHENTICATION SCHEMES USING TRAPDOOR HASH FUNCTIONS

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Santosh Chandrasekhar

Lexington, Kentucky

Director: Dr. Mukesh Singhal

Lexington, Kentucky

2011

ABSTRACT OF DISSERTATION

CONSTRUCTION OF EFFICIENT AUTHENTICATION SCHEMES USING TRAPDOOR HASH FUNCTIONS

In large-scale distributed systems, where adversarial attacks can have widespread impact, authentication provides protection from threats involving impersonation of entities and tampering of data. Practical solutions to authentication problems in distributed systems must meet specific constraints of the target system, and provide a reasonable balance between security and cost. The goal of this dissertation is to address the problem of building practical and efficient authentication mechanisms to secure distributed applications. This dissertation presents techniques to construct efficient digital signature schemes using trapdoor hash functions for various distributed applications. Trapdoor hash functions are collision-resistant hash functions associated with a secret trapdoor key that allows the key-holder to find collisions between hashes of different messages. The main contributions of this dissertation are as follows:

1. A common problem with conventional trapdoor hash functions is that revealing a collision producing message pair allows an entity to compute additional collisions without knowledge of the trapdoor key. To overcome this problem, we design an efficient trapdoor hash function that prevents all entities except the trapdoor key-holder from computing collisions regardless of whether collision producing message pairs are revealed by the key-holder.

2. We design a technique to construct efficient proxy signatures using trapdoor hash functions to authenticate and authorize agents acting on behalf of users in agent-based computing systems. Our technique provides agent authentication, assurance of agreement between delegator and agent, security without relying on secure communication channels and control over an agent's capabilities.

3. We develop a trapdoor hash-based signature amortization technique for authenticating real-time, delay-sensitive streams. Our technique provides independent verifiability of blocks comprising a stream, minimizes sender-side and receiver-side delays, minimizes communication overhead, and avoids transmission of redundant information.

4. We demonstrate the practical efficacy of our trapdoor hash-based techniques for signature amortization and proxy signature construction by presenting discrete log-

based instantiations of the generic techniques that are efficient to compute, and produce short signatures.

Our detailed performance analyses demonstrate that the proposed schemes outperform existing schemes in computation cost and signature size. We also present proofs for security of the proposed discrete-log based instantiations against forgery attacks under the discrete-log assumption.

Keywords: Authentication, digital signature, proxy signature, signature amortization, trapdoor hash functions

Santosh Chandrasekhar

May 3, 2011

CONSTRUCTION OF EFFICIENT AUTHENTICATION SCHEMES USING TRAPDOOR HASH FUNCTIONS

By

Santosh Chandrasekhar

Dr. Mukesh Singhal

Director of Dissertation

Dr. Raphael A. Finkel

Director of Graduate Studies

May 3, 2011

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgements.

Extensive copying or publication of the dissertation in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this dissertation for use by its patrons is expected to secure the signature of each user.

Name	Date

DISSERTATION

Santosh Chandrasekhar

The Graduate School
University of Kentucky
2011

CONSTRUCTION OF EFFICIENT AUTHENTICATION SCHEMES USING TRAPDOOR HASH FUNCTIONS

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Santosh Chandrasekhar

Lexington, Kentucky

Director: Dr. Mukesh Singhal

Lexington, Kentucky

2011

DEDICATION

Dedicated to my mother, Meenakshi Chandrasekhar, my father, Air Commodore C. D. Chandrasekhar (1944 - 1997) and my brother, Anand Chandrasekhar (1978 - 1996).

ACKNOWLEDGMENTS

I thank my advisor, Dr. Mukesh Singhal, for his guidance, support, counseling and technical advice throughout my graduate tenure. I thank Dr. Kenneth L. Calvert for his advice, support and constructive comments that helped me with both technical and personal challenges. Thanks to Dr. Dakshnamoorthy Manivannan, Dr. Uwe Nagel and Dr. Ting-Wen Wu for serving on my dissertation committee, and providing critical and constructive comments on my research work and dissertation. I thank Dr. Raphael A. Finkel for his support as the Director of Graduate Studies and for his corrections to my dissertation. I thank the remaining computer science faculty at the University of Kentucky for helping me build a strong academic foundation and prepare myself for future endeavors.

I extend a special token of appreciation to Saikat Chakrabarti for his support, guidance and friendship that provided focus to my research work and helped shape my Ph.D tenure. Saikat, I thoroughly enjoyed our working relationship and I am glad for our continuing friendship. Thanks to Lei Zhu and Ashley Ritchie, for their support, advice and friendship during my Ph.D. tenure: it meant a lot to me and helped me get through some tough times.

My mother's blessings, her unconditional love and support, her immense sacrifices, her insightful advice and teachings are the reason for all my achievements. My father and brother showered me with love and affection, and instilled values in me that will always dictate my individuality. Their presence has been and always will be a guiding force behind shaping my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation presents techniques for constructing efficient and scalable digital signature schemes using trapdoor hash functions [3, 4, 26, 44, 74] that are designed for providing authentication in emerging large-scale distributed systems.

A distributed system consists of multiple autonomous processing and storage elements that communicate through any form of communication network to provide a common service that covers all aspects of computing and information access. Distributed systems are designed for a myriad of purposes that can include providing resource access to a large number of clients in a transparent, open and scalable manner, and for processing complex and resource intensive programs in an efficient and cost-effective manner. The commercialization of the Internet and the subsequent development of new distributed computing technologies have revolutionized the manner in which governments, business organizations and academic institutions operate and offer their services to clients. Today emerging distributed systems like storage area networks, Cloud computing, and content delivery networks (CDNs) are at the forefront of technologies that are transforming the paradigm of how information is accessed, processed and stored.

Existing technologies like CDN and Cloud computing continue to evolve. We can attribute the evolution of these systems to two lines of research: (1) development of new frameworks and architectures for improving utilization and efficiency; and (2) designing new or transforming existing applications that can benefit from deployment in these emerging systems. Part of the research effort that is driving the evolution of distributed computing technologies is in the field of security. Large-scale distributed systems that are designed to provide services to a large group of users also provide a platform for adversarial users to launch a myriad of attacks with widespread consequences, impacting a large number of components and clients. Moreover, as the number of clients that use distributed computing systems for sensitive tasks (like Internet-based monetary transactions) continue to grow, the cost of attacks due to lack of security measures is increasing as well. Researchers have identified several security threats in systems like CDN and Cloud computing that must be protected against to avoid the large costs of successful attacks. For instance, new applications and architectures for Cloud computing need to address security issues related

to isolation management, multi-tenancy, data exposure, virtual OS security, encryption of data at rest and in transit, service level agreements (SLAs) to obtain Cloud security, trust and compliance issues, among several other security requirements.

Security in distributed systems is a broad area of research that entails authentication, authorization, confidentiality, integrity, and availability. Authentication forms an important part of the spectrum of solutions for protecting network-based operations as well as supporting underlying infrastructure. Authentication allows an entity to establish the legitimacy of a claimed property to another entity. Authentication helps prevent impersonation attacks where an adversary masquerades as a victim's intended communication partner to send malicious data to or receive sensitive information from the victim, and allows the victim to detect malicious tampering of data sent from the intended source. Authentication schemes can provide protection against identity theft or privacy breaches, malware distribution, routing disruptions and copyright infringements, among many other threats. We can use several well-known mechanisms to provide authentication with varying degree of security, performance and features which include password-based techniques, message authentication codes [30], integrity codes [18] and digital signatures [77]. Digital signatures are public key cryptographic schemes that are primarily used for message authentication, and also provide non-repudiation and integrity assurance. Digital signature is among the most popular technique for providing authentication in current computer systems. We can attribute the popularity of digital signatures to their use in the generation of digital certificates, an important component of a public key infrastructure, that are extensively employed to provide entity authentication for organizations wishing to provide secure access to their web servers. Digital signatures also play an important role in authenticating electronic mail, software distribution, financial transactions, and electronic documents of legal significance.

Conventional digital signature schemes, like RSA and DSA [77], are designed to authenticate a single message exchanged between two communicating parties. In distributed systems, data can be stored, processed and routed among multiple locations during the process of an application workflow. In many distributed applications, using conventional digital signatures to authenticate messages can lead to high computation and communication overhead. Examples include applications where the same message is manipulated by multiple entities (e.g., routing using the Border Gateway Protocol), a large number of messages need to be processed by a single entity (e.g., processing feedback in multicast applications) or only a portion of the message that needs to be authenticated is available at the sender at the time of transmission (e.g., distribution of real-time content). A

commercially viable security solution must achieve a reasonable balance between security and cost. The design and construction of authentication schemes must take into account the architecture of the target system and must be tailored towards meeting the specific constraints of the target system to provide a good balance between security and cost. The design of digital signatures must also continuously evolve to tackle new security threats in emerging design and applications of modern distributed systems.

1.1 Research motivation

Cryptographic hash functions are an important building block for a large number of security schemes. Cryptographic hash functions are extensively used for the following purposes: (1) Construction of message authentication codes that can be used for providing data integrity, symmetric data origin authentication, and identification in symmetric-key schemes; (2) Securing password-based authentication protocols from exposure and dictionary attacks; (3) Construction of key derivation function for generating secret keys in key agreement protocols, like Secure Sockets Layer, and; (4) Generation of universally unique identifier that is used to uniquely identify information in distributed systems.

Digital signature schemes employ hash functions for message compression, providing message integrity, non-repudiation and to prevent forgery [60]. In most digital signature schemes, the first step before signing a message is to compute a hash of the message. Next, the hash value of the message is signed in place of the original message. This technique is called the hash-then-sign paradigm. Digital signature schemes require the hash function to be collision resistant, pre-image resistant and second pre-image resistant. Replacing the standard cryptographic hash function with a different cryptographic primitive during generation of digital signatures can lead to new forms of signature schemes with desirable properties. For instance, by generating a signature on a redundancy function of the message, we obtain a signature scheme providing message recovery that can be used to eliminate the communication cost of transmitting the message along with the signature. By generating a signature on a message encrypted using the receiver's public key, we obtain a signcryption scheme that can be used for authenticated transfer of encrypted messages. We use this "hash-replacement" technique as our motivation to develop new forms of digital signature schemes that use trapdoor hash functions instead of conventional cryptographic hash functions to compute the message digest before signing.

Trapdoor hash functions are collision-resistant hash functions associated with a special trapdoor key that enables the possessor of the key to find collisions between hashes of

different messages. Informally, a (trapdoor) collision occurs when two distinct messages have the same (trapdoor) hash value, in which case, the messages are called a collision-producing message pair. A trapdoor hash function is associated to a (private, public) key pair, also referred to as a (trapdoor, hash) key pair. An entity computes trapdoor hash value of a message using the hash key. Collisions are computationally infeasible to find (or compute) without the knowledge of the trapdoor key. However, given the trapdoor key along with the trapdoor hash on a message, it is feasible to find a collision with another given message. When a trapdoor hash function is used within a hash-then-sign signature scheme, it permits the party with knowledge of the trapdoor to *re-use* the signature value to authenticate other messages of choice by finding collisions between the hash of the original signed-message and the new message that needs to be signed. Depending on the efficiency of the trapdoor hash function, this technique of authenticating messages can offer several potential benefits: (1) The computation cost of generating a signature can be reduced to the cost of finding collisions; (2) The cost of verifying a signature can be reduced to comparing two hash values; (3) The communication cost of transmitting authenticating information pertaining to a message can be reduced to the size of the trapdoor hash value. We experiment with various techniques of using the collision finding property of trapdoor hash functions to develop authentication schemes that offer performance advantages over existing schemes while providing security guarantees against forgery attacks.

1.2 Thesis contributions

In this dissertation we introduce new techniques to construct various forms of digital signature schemes using trapdoor hash functions that are suitable for solving emerging security issues related to authentication in modern distributed systems. The general idea behind our techniques is that, rather than generating conventional signatures on a message using a secret key, we use the process of computing hash collisions using a trapdoor key to authenticate messages. Our contributions include the following: (1) We present generic techniques to construct proxy signatures and stream authentication schemes using trapdoor hash functions. Our proposed generic techniques allow the choice of cryptosystem and primitives for implementation, open to future improvements; (2) We present instantiations of our generic constructions using primitives from the discrete log-based cryptosystem to demonstrate the efficacy of the proposed techniques in building practical instances; (3) We evaluate the performance of the proposed discrete log-based signature constructions in terms of communication, computation and storage costs and compare it with existing schemes in

the literature; (4) We conduct an extensive analysis of the security of the proposed signature schemes where we examine the most general mathematical assumptions that are conjectured to be hard, like the discrete log (DL) assumption, and aim to deduce relations between such assumptions and hardness of forging the proposed signatures. Next we elaborate on the specific contributions of this dissertation.

1.2.1 Building an efficient key-exposure-resistant trapdoor hash function

Trapdoor hash functions have been primarily used in the past to construct non-interactive non-transferable signature schemes called *chameleon* signatures [44]. Chameleon signatures are verifiable by no one other than the intended recipient. The general idea behind chameleon signatures is to have the signer compute the trapdoor hash of a message m using the hash key of the recipient and sign the resulting digest. If the recipient uses its trapdoor key and computes a hash collision between the original signed message m and a second message m' to obtain a forged signature on m', to provide non-repudiation, the signer must be able to demonstrate this forgery to a third party. This is typically done by the signer revealing a third message, m'' that has the same trapdoor hash value as m' (or m). The ability of a signer to demonstrate forgery by a receiver requires a trapdoor hashing scheme that allows an entity to compute a trapdoor hash collision with a third message given two messages that result in a collision of their respective trapdoor hash values. However, this property of trapdoor hash value is undesirable when used for authenticating new messages as we explain next.

In this dissertation, we present new techniques to build various forms of digital signatures using trapdoor hash functions. We base our constructions on a central idea that given a message, m, a trapdoor hash of m computed using the signer's hash key, and a signature σ on the trapdoor hash of m computed using signer's secret (signing) key, the signer can authenticate a new message m' by computing a hash collision between m and m' using its (secret) trapdoor key. Any entity that has previously verified σ on m and wishes to check the authenticity of m', simply checks whether the trapdoor hash values of m and m' are equal. This collision based technique to authenticate messages requires that given m and m', it is computationally infeasible for the receiver, or any other third party, to compute m'' that has the same trapdoor hash value as m' (and consequently, m as well) — existing trapdoor hashing schemes [3, 4, 26, 44, 74] were not designed to provide this requirement.

We present a new discrete log-based trapdoor hash function that possesses the required security and performance characteristics to serve as a building block in the construction of

proposed digital signatures schemes. The proposed trapdoor hash function uses ephemeral keys for collision computation and never allows a third party to compute additional trapdoor hash collisions given collision producing message pairs. We provide formal definitions of trapdoor hash functions and their security properties for schemes that use ephemeral keys for collision computation. We present a detailed theoretical analysis, including correctness, security and performance, of the proposed trapdoor hashing scheme.

1.2.2 Proxy signatures for authenticating agents in agent-based computing systems

Today, many large-scale and complex distributed systems like peer-to-peer networks, pervasive and ubiquitous computing environments, Cloud systems, and Grids are highly dynamic, decentralized and loosely coupled with constituent components that are spread throughout a network. Researchers have advocated agent-based computing as the natural computational model for such systems [40]. Agents are software instances that can be delegated by a user to carry out operations on its behalf. However, use of agent-based computing in multi-Cloud, Grid and other distributed systems opens up new avenues for adversarial users to compromise the system. Security in Clouds and other computing systems heavily rely on establishing trust relationships among the actors involved. In agent-based computing, users have to establish trust relationships with agents which includes their security, reliability, availability, and business continuity guarantees. Moreover, sensitive information stored and processed by agents need to be protected from exposure, alteration and corruption. We study the problem of providing efficient authentication of agents in agent-based computing environments.

Proxy signatures have found extensive use in providing message authentication for agents acting on behalf of users in applications such as Grid computing, communications systems, personal digital assistants, information management, and e-commerce. Importance of proxy signatures has been repeatedly highlighted by applied cryptographers through different variations, namely, threshold proxy signatures, blind proxy signatures and so forth. Unfortunately, most recent constructions of proxy signatures only improve on minor weaknesses of previously built schemes, and most often do not deliver formal security guarantees.

We propose a technique to construct provably secure proxy signature schemes using trapdoor hash functions that can be used to authenticate and authorize agents acting on behalf of users in agent-based computing systems. We demonstrate the effectiveness of our approach for creating practical instances by constructing a discrete log-based instantiation of

the proposed generic technique that achieves superior performance in terms of verification overhead and signature size compared to existing proxy signature schemes. We provide formal definitions, security specifications, and a detailed theoretical analysis, including correctness, security and performance, of the proposed proxy signature scheme.

1.2.3 A trapdoor hash based mechanism for stream authentication

Large number of activities in the Internet, like distribution of digital audio, video, software, games, stock quotes and live news feeds, involve transmission of digital streams. In such applications, the end-user's tolerance for high latency, low data rates and playback interruption is small. This means individual components of the stream (like audio or video frames) need to be processed upon reception, with minimal delay, before the entire stream is received. To protect such delay-sensitive digital streams against malicious attacks, security mechanisms need to be designed to process long sequence of bits in an efficient manner that allows the receiver to verify the security of individual components of the stream without excessive delays. This is typically done by dividing the stream into smaller blocks and using a security mechanism to secure each block comprising the stream.

We study the problem of efficient authentication for real-time and delay-sensitive data streams commonly seen in content distribution, multicast and peer-to-peer networks. We propose a novel signature amortization technique based on trapdoor hash functions for authenticating individual data blocks that comprise a digital stream. Our technique represents a radical departure from traditional approaches for amortizing signatures and provides the following features: (1) Resilience against arbitrary transmission losses of intermediate blocks in the stream without affecting verifiability of remaining blocks; (2) Small and constant memory/compute requirements at the sender and receiver for authenticating a stream; (3) Minimal constant communication overhead needed for transmission of authenticating information along with the stream; (4) No wasted bandwidth due to redundant transmission of authenticating information. Our proposed technique makes authentication of streams practical and efficient. We substantiate this claim by constructing a discrete-log based instantiation of the proposed technique. The discrete log-based scheme provides adaptive stream verification, where the receiver has control over modulating computation cost versus buffer size. Our performance analysis demonstrates that the proposed discrete log-based scheme incurs the least per-block communication and signature generation overheads compared to existing schemes with comparable features.

1.3 Thesis organization

In Chapter 2, we provide description of common cryptographic concepts and terminologies relevant to the thesis. We present background information on discrete log-based digital signatures and trapdoor hash functions that are used as building blocks to construct the proposed authentication schemes. We discuss related research on applications of trapdoor hash functions, proxy signatures and stream authentication using digital signatures. In Chapter 3, we present a novel discrete log-based trapdoor hashing scheme that provides all necessary security and performance properties that make it suitable for use as a building block in the construction of the proposed signature schemes. Chapter 4 presents a technique to construct proxy signatures using trapdoor hash functions along with a discrete log-based instantiation of the proposed technique that can be used for providing efficient authentication of agents in agent-based computing systems. Chapter 5 presents a technique for signature amortization using trapdoor hash functions followed by a discrete log-based instantiation of the proposed technique that can be used for providing efficient authentication of real-time and delay-sensitive streams. Chapter 6 concludes the dissertation with a summary of key research contributions and possible research directions in the area of efficient authentication protocols.

Chapter 2

Background and related work

In this chapter, we present a brief introduction to common cryptographic concepts and terminologies, provide a technical description of trapdoor hash functions and digital signatures along with their security properties. We also discuss related research work on application of trapdoor hash functions, development of proxy signatures, and flow authentication mechanisms.

2.1 Cryptographic concepts and terminologies

Authentication: Authentication is the process by which an entity establishes a claimed property to another entity. For example, an entity claiming a legitimate use of a service, or an entry to a system, can establish the claimed legitimacy through use of authentication. The notion of authentication can be further categorized into entity authentication and message authentication.

Entity authentication is a process of verifying that an entity's identity is as claimed through acquisition of some evidence and, that the authenticated entity was active at the time the evidence was created or acquired. A successful entity authentication results in a belief held by the verifier that the authenticated entity possesses the claimed identity.

Message authentication (or data-origin authentication) is the process of verifying whether a message is from a purported source and involves identifying the source of a message. If the message has been modified by an entity which is not the purported source, message authentication should fail. Thus, message authentication provides implicit message integrity. Authentication is based on the possession of some secret information which is known only to the entities participating in the authentication. The basis of authentication can be classified into the following categories: (1) Something you know, like passwords and cryptographic keys; (2) Something you are, like biometric retinal scans and fingerprints, and; (3) Something you posses, like smartcards and physical keys.

One-way trapdoor function: A one-way trapdoor function, $f_k : \mathcal{D} \mapsto \mathcal{R}$ is a function which is easy to evaluate for all $x \in \mathcal{D}$ and difficult to invert for almost all values in $\mathcal{R}$. However, if the trapdoor information k is known, then for all values $y \in \mathcal{R}$, it is easy to compute $x \in \mathcal{D}$ satisfying $y = f_k(x)$.

Negligible function: A negligible function diminishes to 0 faster than the reciprocal of any polynomial. In cryptography, the security of a scheme is judged according to whether the probability of security failure of the scheme is negligible in terms of the cryptographic key length. A function $f(n) : \mathbb{N} \mapsto \mathbb{R}$ is said to be a negligible quantity in n if, for any polynomial $p(n)$, there exists a natural number n_0 such that, for all $n > n_0$, $\frac{1}{f(n)} > p(n)$, i.e., reciprocal of $f(n)$ is unbounded by any polynomial function of n.

Probabilistic Polynomial Time: Deterministic algorithms follow the same sequence of operations each time they execute with the same input. By contrast, randomized algorithms make random decisions at certain points during their execution; hence their execution paths may differ each time they execute with the same input. The random decisions are based upon the outcome of a random number generator. While some random moves lead to a correct result, others lead to incorrect results. A PPT algorithm is a randomized algorithm with bounded error probability whose running time for each input is a polynomial function of the input size. The error probability is determined from its input of randomness.

Computational infeasibility: A computationally feasible problem is a problem that can be solved by an efficient algorithm, i.e., a deterministic or randomized algorithm with execution time expressed as a polynomial function of the input size. Computationally feasible problems can be solved using resources that are manageable, even if the size of the problem is large. All problems that are not computationally feasible are called computationally infeasible. An entity, armed with a polynomial-time algorithm, cannot solve most instances of a computationally infeasible problem with manageable resources. An efficient algorithm for solving a computationally feasible problem need not be efficient in a practical sense because the degree of the polynomial that bounds its time complexity is simply too large. Also, algorithms with non-polynomial time complexities can be useful for solving small instances of computationally infeasible problems efficiently. Practically efficient algorithms are polynomial-time algorithms where the polynomials have very small degrees. In this dissertation, we sometimes

use the term "efficient" to mean "computable (or solvable) by a practically efficient algorithm".

Provable Security: Cryptographic protocols or schemes are built using atomic primitives, like Advanced Encryption Standard (AES) or one-way functions, that are based on mathematical problems conjectured to be hard. Security assurances provided by cryptographic protocols rely on the confidence in the security of the underlying primitives. A limited viewpoint in cryptography regards the security of a cryptographic protocol to be equivalent to solving the underlying mathematical problem. However, many attacks on cryptographic schemes succeed, not by cryptanalyzing the underlying primitive, but rather by finding a weakness in the protocol design [10, 12].

The reductionist security argument provides a means to researchers to, in a sense, "provably" demonstrate the security of a cryptographic protocol. The idea of a reductionist security argument is as follows: Assume that the goal is to provide authentication via digital signatures. The first step is to describe a formal adversarial model and define what it means for a signature scheme to be secure. Based on the model and security definition, a particular scheme can be analyzed against attacks from the point of view of satisfying the definition. Eventually, one proves that the scheme is secure via a reduction showing that anyone who has an algorithm to mount a successful attack on the signature scheme, can use the same algorithm to solve the underlying mathematical problem (that is used as the basis for the claimed security of the signature scheme) with relatively little additional effort. This type of security proof allows cryptanalysts to focus on studying the security of the atomic primitive (or the hardness of underlying mathematical problem) rather than directly cryptanalyzing a security protocol — if cryptanalysts were to find a weakness in the protocol, they would have discovered a weakness in the underlying atomic primitive. Moreover, if the underlying atomic primitive is known to be secure (or the underlying mathematical problem is hard), we can deduce, without further cryptanalysis, that the protocol is secure. For a reductionist security argument to hold, a formal notion of security of the underlying atomic primitive is required.

Random Oracle Model: A central step in proving the security of a cryptographic protocol using the reductionist security argument is to describe an adversarial model. An adversarial model describes the means, capabilities and goals of an adversary acting against the target cryptographic protocol. Important components

of an adversarial model include abstraction of the communication channels and formalization of mathematical objects, like infinite random strings and oracles. Models used for proving the security of a cryptographic protocol include the standard model, generic group model and the random oracle model. The standard model does not use any special mathematical objects and assumes existence of a reliable but insecure communication channel. Proofs in the standard model are typically unappealing [63] and lead to protocol constructions that are inefficient in practise [10]. The generic group model assumes that non-trivial properties of the representation of the elements of an algebraic structure (e.g. a group) under consideration cannot be exploited [58]. The generic group model provides the adversary with a random encoding of the algebraic structure and oracle to answer permissible operations in the algebraic structure [76]. Generic group models are typically used for computing the lower bounds on the complexity of mathematical problems conjectured to be hard (like the discrete log problem) that form the basis of several cryptosystems using algorithms that only make use of operations in the algebraic structure and do not consider the encoding of the elements of the algebraic structure.

The random oracle model is a popular choice in the security research community to obtain security results for many efficient and practical schemes that are constructed using hash functions. In the random oracle model, the behavior of a hash function is emulated by a deterministic and efficient function that outputs uniformly distributed random values. All entities including the adversary, are given access to the random oracle $\mathcal{O}_H$ that behaves as follows: The oracle $\mathcal{O}_H$ initializes an empty list L. The list L stores tuples of the form (m_i, h_i), where $m_i \in \{0,1\}^*$ are queries and $h_i \in \{0,1\}^n$ are random answers. For efficiency the list L can be sorted by m_i. When an entity queries $\mathcal{O}_H$ with a value m_i, the oracle searches the list L for the tuple (m_i, h_i). If the oracle finds the tuple, $\mathcal{O}_H$ returns h_i as the answer. If not, the oracle chooses a random value $h_i \in \{0,1\}^n$, stores (m_i, h_i) (at an appropriate location) in the list L and returns h_i as the answer. Random oracles do not exist in practise. In the real-world, hash functions only emulate the random oracle behavior to a precision where the difference is preferably a negligible quantity. However, if hash functions used in a cryptographic scheme or protocol have no "obvious" flaw, then a security proof for such a scheme using their idealized version can be considered as a useful test-bed for checking the security of the scheme or protocol. Today, designing a cryptographic scheme so that it is argued to be secure in the random oracle model is widely accepted

as a good engineering principle.

2.2 Digital signatures

A digital signature is a cryptographic mechanism to provide authentication, message integrity and non-repudiation. The process of signing entails transforming the message and some secret information held by the entity into a *tag*, called a signature. This signature provides a means for an entity to bind some information regarding itself (e.g., its identity) to the message that is signed, and prove ownership of a message to another entity. A signature is authentic if it was indeed created by the specified entity.

Most digital signature schemes employ hash functions for message compression, providing message integrity, non-repudiation and for preventing forgery [60]. Formally a digital signature scheme can be defined as follows:

Definition 2.2.1. *A digital signature scheme* `DS` *is the tuple* ⟨`ParGen`, `KeyGen`, `SigGen`, `SigVer`⟩ *whose components are defined as follows:*

`ParGen`: *A PPT algorithm that takes a security parameter* λ *as input and outputs system public parameters* params.

`KeyGen`: *A PPT algorithm that takes system parameters* params *as input and outputs a (private, public) key pair* (SK, PK).

`SigGen`: *A PPT algorithm that takes system parameters* params, *a message* m *and a secret key* SK *as inputs and outputs a signature* σ *on* m.

`SigVer`: *A deterministic algorithm that takes system parameters* params, *public key* PK, *message* m *and a candidate signature* σ *on* m *as inputs and outputs* Valid *or* Invalid.

A signature σ on a message m signed using SK is said to be authentic or valid under PK if it passes the verification procedure, i.e., `SigVer` with params, σ, m and PK as inputs, outputs *Valid*. We say that (m, σ) is a valid (message, signature) pair under public key PK.

2.2.1 Security of digital signatures

Let (SK, PK) represent the (private, public) key pair of an arbitrary entity E. A forged signature σ on a message m is one that is valid under public key PK where E never signed the message m. The goal of an adversary $\mathcal{A}$ is to forge signatures that appear to be generated by some other entity E. Security of digital signatures is judged by their ability

to resist signature forgery by relying on the difficulty of solving some well-known problem such as the DL problem and the integer factorization problem.

A forgery can be either a total break, selective forgery or an existential forgery. A total break occurs when an adversary is able to compute the private key of the signer. Selective forgery allows an adversary to create a valid signature on a message chosen by or known to the adversary beforehand. Existential forgery occurs when an adversary is able to forge a signature on some message, where the adversary has little or no control over the message on which the signature forgery is obtained.

An adversary can launch the following two-types of attacks to forge signatures:

1. Key-only attack: An adversary knows only the signers public key PK.

2. Message attack: An adversary is able to obtain valid (message, signature) pairs for a set of messages that are known to or chosen by the adversary. Message attacks can be further classified into the following three categories:

   (a) Known-message attack: An adversary obtains a set of valid (message, signature) pairs, where the messages are known to the adversary but not chosen by it.

   (b) Chosen-message attack: An adversary obtains a set of valid (message, signature) pairs, where the messages are chosen by the adversary before any signatures are obtained.

   (c) Adaptive chosen-message attack: An adversary is allowed to use the signer as an oracle and request signatures on messages which are chosen by the adversary. The choice of messages can depend on the signer's public key or previous signed messages.

In principle, an adaptive chosen-message attack, where the adversary attempts to forge a signature by using the signer as an oracle and requesting signatures on messages of choice, is the most difficult to prevent. Advantage of an adversary in forging a signature is the probability that the adversary outputs a valid forgery. A signature scheme is said to be secure if there exists no PPT forger that breaks the scheme with non-negligible advantage. A detailed discussion on the types of attacks on digital signatures and their security considerations can be found in [60].

2.2.2 Discrete log-based signatures

Primitives from DL-based cryptosystems have been extensively employed in the construction of a large number of security protocols. These include the ElGamal [32] signature schemes

and its variants [38], ElGamal encryption [32] and Diffie Hellman [27] key exchange and its derivatives. The security of these protocols rely on the intractability of the discrete log problem. The DL problem and assumption can be defined as follows:

Definition 2.2.2. *Let p and q be primes such that $q \mid (p-1)$, whose sizes are determined by a security parameter λ. Let α be an element of order q in $\mathbb{Z}_p^*$ that generates the unique cyclic subgroup G of $\mathbb{Z}_p^*$ of order q. The DL Problem in G can be defined as follows: Given $(p, q, \alpha \in \mathbb{Z}_p^*, \beta \in G)$, find an index k, $0 \leq k \leq q-1$, such that $\beta = \alpha^k \mod p$ or determine that there is no such index.*

Let $\mathcal{A}$ be a PPT algorithm that solves the DL problem in G. Define the advantage of the DL solver $\mathcal{A}$ as: $\mathsf{Adv}_{\mathcal{A}}^{DL}(\lambda) = Pr[\mathcal{A}(p, q, \alpha, \beta) = k \mid \alpha \in_R \mathbb{Z}_p^*, 0 \leq k \leq q-1, \beta = \alpha^k \mod p]$, where the notation $\in_R$ denotes "randomly chosen from". The probability is over the random choices of α, k, the size of security parameter λ, and the input of randomness for the algorithm $\mathcal{A}$.

DL Assumption*: The cyclic subgroup G satisfies the DL Assumption if $\mathsf{Adv}_{\mathcal{A}}^{DL}(\lambda)$ is a negligible function.*

The ElGamal family of signatures continue to be one of the most efficient signature schemes in terms of storage overhead, signature size and computation cost. Moreover, the National Institute of Standards and Technology (NIST) adopted a specific variant of ElGamal (DSA) for use in their Digital Signature Standard (DSS), specified in FIPS 186 and adopted in 1993 [77]. Other popular signature schemes within this family include the provably secure Schnorr variant [72], and Nyberg-Rueppel family of signature schemes with message recovery [64].

System public parameters, common to all variants of ElGamal, are chosen as $\mathsf{params} = \langle p, q, \alpha \rangle$, where p and q are 1024-bit and 160-bit primes, respectively, $q \mid p-1$, and α is an element of order q in $\mathbb{Z}_p^*$. The long-term (private, public) key pair of an entity is given by (x, X), where $x \in_R \mathbb{Z}_q^*$ and $X = \alpha^x \in \mathbb{Z}_p^*$. To sign a message m, an entity first chooses an ephemeral private key $k \in_R \mathbb{Z}_q^*$ and computes ephemeral public key $r = \alpha^k \mod q$. Next, the entity computes the *signature value* t by solving a signing equation that can be expressed in the generalized form as:

$$\pm A \equiv \pm xB \pm kC \mod q$$

The quantities, A, B and C permute over $H(m)$, r and t individually, or over functions $f(\cdot,\cdot)$, $g(\cdot,\cdot)$ of (m, r), (m, t) and (r, t), and unity, where, $H : \{0,1\}^* \mapsto \mathbb{Z}_q^*$ is a cryptographic hash function. The parameter t together with r constitute the resulting signature $\sigma = \langle t, r \rangle$

on m. The signature σ on m can be verified under public key X by checking whether the equivalence, $\alpha^A \equiv X^B r^C \mod p$, holds. The functions $f : \{0,1\}^* \times \mathbb{Z}_q^* \mapsto \mathbb{Z}_q^*$ and $g : \mathbb{Z}_q^* \times \mathbb{Z}_q^* \mapsto \mathbb{Z}_q^*$ are carefully chosen so that the signing equation can be solved for t, the essence of the signature. For the DSA variant, $A = H(m)$, $B = r$, $C = t$, and verification involves checking whether $\alpha^{H(m)} \stackrel{?}{\equiv} X^r r^t \mod q$ holds.

In this dissertation we present instantiations of various forms of trapdoor hash-based signature schemes using a particular variant of the ElGamal family of signatures, called the Schnorr signature scheme [72]. Denoted as `DL-Schnorr`, the Schnorr signature scheme can be described as follows:

`DL-Schnorr.ParGen:` Entities choose and agree upon common system public parameters $\mathsf{params} = \langle p, q, \alpha, H \rangle$, where p and q are 1024-bit and 160-bit primes, respectively, $q \mid p-1$, α is an element of order q in $\mathbb{Z}_p^*$ and $H : \{0,1\}^* \mapsto \mathbb{Z}_q^*$ is a cryptographic hash function.

`DL-Schnorr.KeyGen:` An entity uses the system public parameters params, to generate its private and public key pair, $(SK, PK) = (x, X)$, where $x \in_R \mathbb{Z}_q^*$ and $X = \alpha^x \in \mathbb{Z}_p^*$.

`DL-Schnorr.SigGen:` Given a message $m \in 0, 1^*$, an entity uses its private key $SK = x$ and executes the following:

1. Choose an ephemeral private key $k \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral public key as $r = \alpha^k \mod q$.
2. Solve for the *signature value* t in the equation, $t \equiv xH(m||r) + k \mod q$, where $m||r$ denotes the concatenation of r to m.

The entity outputs $\sigma = \langle t, r \rangle$ as the resulting signature on message, m.

`DL-Schnorr.SigVer:` To verify a signature σ on m under public key X, an entity executes the following:

1. Parse σ as the tuple $\langle t, r \rangle$ and compute $h = H(m||r)$.
2. Compute $r' = \alpha^t X^{-h} \mod q$.

If $r \equiv r' \mod q$, then σ is a valid signature on m and the entity outputs $Valid$. Else, verification fails and the entity outputs $Invalid$.

Security of the Schnorr signature scheme against forgery is based on the following well-known theorem [70].

Theorem 2.2.1. *The Schnorr variant [72] of the ElGamal family of signatures is secure against an adaptive chosen message attack in the random oracle model under the discrete logarithm assumption.*

2.2.3 Proxy signatures

The concept of a proxy signature was introduced by Mambo et al. [57]. Proxy delegation is a process by which an entity, the delegator, transfers its signing rights and capabilities to another entity, the proxy. Following delegation, the proxy can generate signatures, called proxy signatures on behalf of the delegator. Mambo et al. [57] classified proxy delegation into partial delegation, full delegation and delegation by warrant, presented possible constructions for proxy signatures, and provided an informal security analysis. Later classifications of proxy signatures included partial delegation by warrant [43], and strong/weak proxy signatures [47].

Since their introduction, researchers have focused on developing new proxy signatures by enhancing the security or efficiency of previous schemes, but only relying on heuristic security arguments [6, 45, 65, 78, 79]. Motivated by this, Boldyreva et al. [13] proposed the first formal security notion for proxy signature schemes and presented a provably secure variant of Kim et al.'s scheme [43]. Later, Lee et al. [48] analyzed the necessity of secure channels in proxy signature schemes like [47, 57, 69, 43]. Malkin et al. [56] extended the model by Boldyreva et al. for one level of delegation, to fully hierarchical proxy signatures and proved the equivalence of proxy signatures and key-insulated signatures [28]. Zhang et al. [83] proposed the first proxy signature scheme based on bilinear pairings in elliptic and hyperelliptic curves. More recently, Schuldt et al. [73] proposed a stronger security model compared to models by Boldyreva et al. [13] and Malkin et al. [56], and presented a generic construction of a provably secure proxy signature scheme based on sequential aggregate signatures in their stronger security model.

Proxy signatures have found extensive use in solving authentication issues in mobile agent applications where autonomous software agents need to migrate across heterogeneous execution environments and reliably perform transactions in electronic commerce [13, 47, 46]. Delegation of rights is also common in providing authorization for delegated entities in distributed systems and for workflow management of information systems [5]. Leiwo et al. [49] proposed a means to authenticate state alterations in distributed shared objects using proxy signatures. Leiwo et al. [49] used the scenario of free software distribution to show how proxy signatures could be used by core group members to sign update

state messages on behalf of the administrator. Ahn et al. [1] proposed integration of a role-based delegation framework [7] in pervasive computing environments to provide selective information sharing while minimizing the risks of unauthorized access. Proxy signatures have potential application in securing such access delegation systems to provide authentication and non-repudiation in addition to access control. Proxy signatures can also be used to provide authentication of user proxies in Grids [31, 80].

2.2.4 Signature-based stream authentication techniques

Conventional techniques for message authentication require the sender and the receiver to have the ability to store the entire message before authenticating the message. However, in many instances, like transmission of audio, video, live news feeds, stock quotes and other forms of digital content, the message is split into smaller blocks, and individual blocks in the stream need to be authenticated upon reception with minimal delay before the entire stream is received [33]. Researchers have proposed several techniques for stream (or flow) authentication that aim at reducing the computation and communication overhead associated with securing individual blocks that comprise a stream. These techniques can be divided into MAC-based schemes like *Timed Efficient Stream Loss-tolerant Authentication* (TESLA) [68] (and its variants) and signature-based schemes like *Efficient Multichained Stream Signature* (EMSS) [68], *Augmented Chain* (AC) graph-based stream authentication [34], *Signature Amortization using Information Dispersal Algorithm* (SAIDA) [66], and *Wong and Lam* (WL) [82]. While TESLA is efficient and robust against data loss, it requires time synchronization between a signer and a verifier, sufficiently large buffers for storing all unverified blocks (until the verification key is received), and storage of long key chains which can lead to scalability issues. This makes TESLA less suitable for authenticating stream, and vulnerable to DoS attacks that cause buffer overflow.

To reduce per-block overhead, signature-based stream authentication techniques either rely on amortizing a single signature over multiple blocks or designing extremely fast signature schemes, like k-time signatures [71] and *Bins and Balls* (BiBa) [67] to sign each block. Designing extremely fast signature schemes often come at the cost of unreasonably high storage and communication overheads [66, 82]. To amortize a signature over multiple blocks in a stream, EMSS and AC use hash chains, SAIDA splits the signature and hash value of each block over multiple blocks, and WL uses Merkle trees. EMSS, AC and SAIDA are probabilistic authentication schemes, i.e., the ability of a receiver to verify a received block depends on whether the receiver has some additional blocks of the stream in its

possession (which inherently requires a verifier to maintain a buffer with multiple data blocks). Thus, the probability that a receiver is able to verify a block depends on the nature (bursty, independent, etc.) and probability of data loss during stream transmission. EMSS and AC rely on redundant placement of multiple hashes in each block to deal with blocks that are lost during transmission. SAIDA relies on erasure codes to recover from losses. In probabilistic authentication schemes, maintaining a reasonable probability of verification in the presence of high data loss leads to higher per-block communication overhead and increased size of verifier-side data buffer. If per-block communication overhead is restricted, verification probability drops with an increase in data loss.

The WL scheme is the only known deterministic stream authentication protocol. In the WL scheme, a stream is divided into segments, with each segment containing multiple blocks (in WL each block is a single packet). The computational overhead at the signer and verifier, the signer's buffer size and the per-block communication overhead are highly dependent on the segment size. Computational overhead at the signer and verifier increases with a decrease in the segment size. On the other hand, the buffer size at the signer and the per-block communication overhead increase with an increase in the segment size. With a reasonable block size (say, 16 [82]), the signer-side delay increases (affecting real-time performance) along with the per-block communication overhead (to a magnitude of 100's of bytes).

Recently, Lysyanskaya et al. [55] proposed a stream authentication technique, *Authenticated Error-Correcting Codes* (AECC) using error correcting codes that is provably secure in a formal adversarial network model that limits the capabilities of an adversary to injecting and deleting packets in discrete quantities. The AECC scheme only requires one signature operation for the entire stream and adds only a constant size authentication overhead per packet. However, the AECC scheme requires the sender to possess the entire stream before signing and thus, cannot be used for real-time generated content.

2.3 Trapdoor hash functions

Trapdoor hash functions are collision-resistant hash functions associated with a special trapdoor key that enables the possessor of the key to find collisions between hashes of different messages. A trapdoor hash function is associated to a (private, public) key pair, also referred to as a (trapdoor, hash) key. Collisions are computationally infeasible to find without the knowledge of the trapdoor key. However, given the trapdoor key along with the trapdoor hash on a message, it is feasible to find a collision. A trapdoor hashing scheme

consists of four algorithms for parameter generation, key pair generation, hash generation and collision computation, respectively. Formally a trapdoor hashing scheme can be defined as follows:

Definition 2.3.1. *A trapdoor hashing scheme,* `TH`*, is a tuple* ⟨`ParGen`, `KeyGen`, `HashGen`, `TrapColGen`⟩ *whose components are defined as follows:*

`ParGen`**:** *A PPT algorithm that takes a security parameter* λ *as input and outputs system public parameters* params*.*

`KeyGen`**:** *A PPT algorithm that takes* params *as input and outputs a (trapdoor, hash) key pair* (TK, HK)*.*

`HashGen`**:** *A PPT algorithm that takes* params*,* HK*, a message* m *and a random element* r *as inputs, and outputs the hash* $TH_{HK}(m, r)$*.*

`TrapColGen`**:** *A PPT algorithm that takes* params*, a pair* (TK, HK)*, a message* m*, a random element* r *and an additional message* $m' \neq m$ *as inputs, and outputs a collision parameter* c *such that* $TH_{HK}(m, r) = TH_{HK}(m', c)$*.*

The function TH is said to be associated with the (long-term) hash key HK. For trapdoor hash function to be practical, computing the digest of a message using the `HashGen` algorithm and collisions using the `TrapColGen` algorithm must be achievable in polynomial time.

2.3.1 Security properties of trapdoor hash functions

Security notions associated with trapdoor hashing schemes include collision forgery resistance and semantic security [74]. Collision forgery resistance implies that given system parameters, params and hash key, HK, it is computationally infeasible to find a tuple $\langle m, m', r, c\rangle$ such that $TH_{HK}(m, r) = TH_{HK}(m', c)$. Semantic security of a trapdoor hashing scheme ensures that for all hash keys HK, and all pairs of messages m and m', where $m \neq m'$, the probability distributions of the hash values $TH_{HK}(m, r)$ and $TH_{HK}(m', r)$ are computationally indistinguishable.

Another well-known property of trapdoor hash function is resistance to key exposure [4]. A trapdoor hashing scheme is said to suffer from the *key exposure problem* if given the system parameters params and the tuple $\langle m, m', r, r', HK\rangle$ such that $TH_{HK}(m, r) = TH_{HK}(m', r')$, the trapdoor key, TK corresponding to the hash key, HK can be computed in polynomial time. The problem of key exposure was first discovered by Ateniese et al. [3]. Ateniese et al.

observed that when a trapdoor hashing scheme is used for constructing non-interactive non-transferable signature scheme, called *chameleon* signatures, vulnerability of the trapdoor hash function to key exposure undermines the property of non-transferability. In Chapter 3, we present further details on the effects of key exposure in trapdoor hash functions on applications of trapdoor hash functions in chameleon, online/offline and other signature schemes.

2.3.2 A discrete log-based trapdoor hash function

Krawczyk et al. [44] proposed a simple DL-based trapdoor hashing scheme, `DL-TH`, which was later used by Shamir et al. [74] in the construction of their online/offline signature scheme. The `ParGen`, `KeyGen`, `HashGen` and `TrapColGen` algorithms of the DL-based hashing scheme are executed as follows:

`DL-TH.ParGen`: Entities choose and agree upon common system public parameters $\mathsf{params} = \langle p, q, \alpha, H \rangle$, where p and q are 1024-bit and 160-bit primes, respectively, $q \mid p-1$, α is an element of order q in $\mathbb{Z}_p^*$ and $H : \{0,1\}^* \mapsto \mathbb{Z}_q^*$ is a cryptographic hash function.

`DL-TH.KeyGen`: An entity uses the system public parameters params, to generate its (private) trapdoor key and (public) hash key pair, $(TK, HK) = (y, Y)$, where $y \in_R \mathbb{Z}_q^*$ and $Y = \alpha^y \in \mathbb{Z}_p^*$.

`DL-TH.HashGen`: An entity generates a trapdoor hash of a message $m \in \mathbb{Z}_q^*$ using the hash key Y, by choosing an element $r \in_R \mathbb{Z}_q^*$ and computing the hash as $TH_Y(m, r) = \alpha^{H(m)} Y^r \bmod q$. The function TH is said to be associated with the hash key Y. Note that the trapdoor hash function on the hashed message m does not affect any security properties [44].

`DL-TH.TrapColGen`: Given the trapdoor key and hash key pair (TK, HK), $m, r \in \mathbb{Z}_q^*$ and an additional message $m' (\neq m) \in \mathbb{Z}_q^*$, an entity computes a collision parameter $c \in \mathbb{Z}_q^*$ such that, $TH_{HK}(m, r) = TH_{HK}(m', c)$, by solving $c \equiv y^{-1}(H(m) - H(m')) + r \bmod q$. Since $m' \neq m$ and r is uniformly distributed in $\mathbb{Z}_q^*$, the computed value $c \neq r$ is unique and is also uniformly distributed in $\mathbb{Z}_q^*$ [74].

The `DL-TH` trapdoor hashing scheme is collision resistant as defined by Shamir et al. [74]. We present a brief proof of resistance of `DL-TH` against collisions.

Theorem 2.3.1. *The trapdoor hashing scheme,* `DL-TH` *is collision resistant.*

Proof: [74] Collision forgery resistance implies that given system parameters params and hash key HK as inputs, a PPT collision forger $\mathcal{F}$ has negligible probability to successfully output the tuple $\langle m, r, m', r' \rangle$ that satisfies $m \neq m'$ and $TH_{HK}(m, r) = TH_{HK}(m', r')$.

To the contrary, assume that there exists a PPT collision forger $\mathcal{F}$ that outputs $\langle m, r, m', r' \rangle$ that satisfies $m \neq m'$ and $\alpha^{H(m)} Y^r \equiv \alpha^{H(m')} Y^{r'} \mod q$ with a probability which is not negligible. The discrete log y of the hash key Y with respect to the basis α can be calculated in polynomial time from the output, as follows: From the equality in the trapdoor hash values of m and m' we know that $H(m) + yr \equiv H(m') + yr' \mod q$. Also, given that $m \neq m'$ and H is a cryptographic hash function, we know that $H(m) \neq H(m')$ and $(r' - r)$ is non-zero modulo q. Thus, the discrete log of Y with respect to the basis α can be computed as $y = (r' - r)^{-1}(H(m) - H(m')) \mod q$. This contradicts the discrete log assumption. □

The trapdoor hashing scheme described above suffers from the well known *key exposure problem* [4]. Given two pairs (m, r) and (m', c) such that $TH_{HK}(m, r) = TH_{HK}(m', c)$, any third party can compute the trapdoor key TK as

$$TK = (H(m) - H(m'))(c - r)^{-1} \mod q \tag{2.1}$$

In Chapter 3 we use the `DL-TH` scheme as a basis for construction of an efficient trapdoor hashing scheme, called `DL-MTH`, that is collision-resistant, key-exposure-resistant and semantically secure. We use the proposed `DL-MTH` scheme as a building block to construct efficient authentication schemes for securing various distributed applications.

2.3.3 Applications of trapdoor hash functions

The concept of a trapdoor hash function was originally derived from the notion of trapdoor commitments proposed by Brassard et al. [15]; Krawczyk et al. [44] used trapdoor hash functions (referred to as chameleon hash) to construct a non-interactive non-transferable signature scheme, called chameleon signatures (closely related to undeniable signatures), under the hash-and-sign paradigm. Chameleon signature allows a signer to undeniably commit to the contents of a signed document, but does not allow the recipient of the signature to disclose the signer's commitment to a third party without the signer's consent. The problem of key-exposure in trapdoor hash functions was first identified by Ateniese et al. [3] who presented a partial solution to the problem by constructing an identity-based trapdoor hash function that uses a different public key to compute the digest of each message. Chen et al. [26] proposed the first construction of a key-exposure free trapdoor hash function in Gap Diffie-Hellman Group with bilinear pairings and described

an application of chameleon signatures in receipt-free electronic voting schemes. Later, Ateniese et al. [4] proposed several constructions of trapdoor hash functions without key exposure, and provided several applications of trapdoor hashing like private auctions and secure software distribution. Shamir et al. [74] employed trapdoor hash functions to develop a new paradigm, called hash-sign-switch, that can be used to convert any signature scheme into an online/offline signature scheme [29]. In online/offline signature schemes, the signature generation procedure is split into two phases that are performed offline (before the message to be signed is known) and online (after the message is known). By shifting the computational burden to the offline phase, online/offline signatures can achieve very high efficiency for signing messages during the online phase.

Mehta et al. [59] introduced the idea of using trapdoor hash functions to build one-time proxy signatures (one proxy signature per delegation) by exploiting the key-exposure property of trapdoor hash functions. In their scheme, the verifier need not be aware of the existence of a proxy. The delegation process is considered confidential and verification of a proxy signature requires only the delegator's public key, and follows standard signature verification procedures. Mehta et al. use techniques inherited from chameleon signatures to resolve disputes between verifier and delegator, and between proxy and delegator.

2.4 Summary

In this chapter, we presented a brief introduction to common cryptographic concepts and terminologies that will aid the reader in understanding the topics covered by this dissertation. We provided a technical description of digital signatures and trapdoor hash functions along with their security properties and a brief review of a well-known discrete log-based signature scheme, `DL-Schnorr` [72] and trapdoor hashing scheme, `DL-TH` [44]. We also discussed related research work on proxy signatures [57] that are used for authenticating agents acting on behalf of users, signature amortization schemes that are used for authenticating individual blocks comprising a data stream and applications of trapdoor hash functions in the construction of chameleon [44], online/offline [29] and proxy signatures.

In the next chapter we present a novel and efficient discrete log-based trapdoor hash function that will be used as a building block to construct various authentication schemes including a proxy signature-based agent authentication scheme, and signature amortization-based stream authentication scheme.

Chapter 3

Building an efficient key-exposure-resistant trapdoor hash function

3.1 Introduction

Trapdoor hash functions were originally designed to construct non-interactive non-transferable signature schemes, called chameleon signatures [44]. The idea is to use a trapdoor hash function to compute the digest h of a message m and sign h using an arbitrary signing algorithm. If the hash key associated to the trapdoor hash function belongs to the recipient, then the signature is verifiable by no one other than the intended recipient. The recipient can obtain a signature on a second message m' by using its trapdoor key to compute a hash collision between the original signed message m and m'. In this case, the signer can prove that the signature on m' is a forgery by revealing the original signed message m with the same hash value as m'. Since it is infeasible for the signer to compute trapdoor hash collisions, revealing m is seen as a proof of forgery by the recipient and revokes the original and forged signatures, thus providing non-repudiation.

The original chameleon signature scheme by Krawczyk et al. [44] uses a trapdoor hash function that is vulnerable to key exposure. Ateniese et al. [3] observed that a signature forgery by the recipient (i.e., revealing a message m' with the same hash value as the original signed message m) results in the signer recovering the trapdoor key of the recipient. Following the recovery of trapdoor key, the signer can invalidate other signatures which were designated to be verified by the corresponding hash key, thus denying other signatures given to the recipient. Due to the potential damage to the recipient resulting from a signature forgery, a third party is likely to believe the authenticity of a recipient's claim. Thus, the threat of key exposure creates a strong disincentive for the recipient to forge signatures, undermining the property of non-transferability provided by the scheme. To fix this issue Ateniese et al. [3] proposed an identity-based trapdoor hashing scheme where the signer uses a different hash key to sign each message. A hash collision produced by the recipient only results in the exposure of trapdoor key associated to the single hash key used for signing a single message, thus preventing the signer from denying signatures on other messages. The aforementioned solution to the key exposure problem is partial, as the scheme still allows the exposure of a single-use trapdoor key. Chen et al. [26] proposed the first key-exposure free

trapdoor hash function in Gap Diffie-Hellman Group with bilinear pairings that prevents exposure of the trapdoor key even if a message pair resulting in a hash collision is revealed. Later, Ateniese et al. [4] presented non-pairing based key-exposure-free trapdoor hashing schemes. Both Chen et al.'s and Ateniese et al.'s schemes [4, 26] prevent key exposure by using an ephemeral component, which we call a transaction identifier, during computation of a message digest that is unique for every message that is hashed. More specifically, in the schemes by Chen et al. and Ateniese et al., the trapdoor hash function TH takes a hash key HK, message m, a random value r and a transaction identifier i as inputs and outputs the digest value $h = TH_{HK}(m, r, i)$. Collisions can only be computed under the same transaction identifier i, i.e., given the tuple $\langle m, r, i \rangle$, an entity with the trapdoor key TK can generate $\langle m', r' \rangle$ such that $TH_{HK}(m, r, i) = TH_{HK}(m', r', i)$.

3.1.1 Problem statement

The constructions of key-exposure-free trapdoor hash functions by Chen et al. [26] and Ateniese et al. [3, 4] are tailored towards addressing the property of non-transferability and message hiding in chameleon signatures. The message hiding property of chameleon signatures ensures that during settlement of disputes between a signer and a recipient to ascertain whether or not a purported message was indeed the original one committed by the signer, if the contested signature is invalid, the signer must be able to produce a message that is different from the original signed message (thus, hiding the original message) and authenticated by the same signature as the one on the purported message. To provide message hiding and non-transferability, majority of the trapdoor hashing schemes by Chen at al. [26] and Ateniese et al. [3, 4] (except one scheme proposed in [4]) require that given two messages that result in a collision of their respective trapdoor hash values under the same transaction identifier, an entity must be able to compute a trapdoor hash collision with a third message under the given transaction identifier, without revealing the secret trapdoor key. More specifically, if the recipient computes a hash collision between the original signed message m and a new message m' such that $h = TH_{HK}(m, r, i) = TH_{HK}(m', r', i)$, where i is the transaction identifier and (r, r') are random values, then the signer can compute a pair $\langle m'', r'' \rangle$ such that $TH_{HK}(m'', r'', i) = h$. This property, although beneficial for chameleon signatures, represents a security flaw when trapdoor hash functions are used to construct signature schemes where, rather than authenticating messages using conventional digital signatures, we use the process of computing hash collisions using a trapdoor key to authenticate messages.

The general idea behind using collision computation for message authentication is as follows: Let A be an entity in possession of a (private, public) key pair, (SK, PK) and a (trapdoor, hash) key pair, (TK, HK). Assume that A generates a signature σ using SK on the trapdoor hash of a message $TH_{HK}(m, r)$ and publishes $\langle \sigma, m, r \rangle$ in a publicly available directory. Any entity that wishes to communicate with A first retrieves $\langle \sigma, m, r \rangle$ and verifies the validity of σ on $TH_{HK}(m, r)$ using HK and PK. Now, when A needs to generate a new signature on a new message m', it computes r' such that $TH_{HK}(m, r) = TH_{HK}(m', r')$ using its trapdoor key TK. The value r' represents a *signature* on m' that can be verified by simply checking whether $TH_{HK}(m, r) = TH_{HK}(m', r')$. In this scheme, the ability of a third party to compute a message m'', r'' such that $TH_{HK}(m, r) = TH_{HK}(m', r') = TH_{HK}(m'', r'')$ results in a forged signature σ on m''. Moreover, we note that using collision computation for message authentication provides performance advantages over the conventional technique of authenticating messages using digital signatures when the cost of computing collisions is cheaper than signature generation and when cost of hash computation is cheaper than signature verification. We observe that the two schemes by Harn et al. [37] and the scheme by Ateniese et al. [4] that do provide the property of preventing additional collisions are computationally more expensive for collision and hash computation than the cost of conventional signature generation and verification, respectively. These inefficiencies of existing trapdoor hashing schemes by Harn et al. and Ateniese et al. undermine our incentive to use the process of computing collisions, instead of conventional signatures, for message authentication. Thus, building authentication schemes that use collision computation for message authentication require a trapdoor hashing scheme that is efficient, key-exposure-free, collision-resistant and *never* allows a third party to compute hash collisions given two messages that hash to the same value.

3.1.2 Contributions

In this chapter, we introduce an efficient, discrete log-based trapdoor hashing scheme that is resistant to collisions and key-exposure. The proposed trapdoor hashing scheme is designed for use as a building block to construct signature schemes that employ the process of computing hash collisions using a trapdoor key to authenticate messages. The proposed scheme achieves key-exposure-freeness by splitting the (trapdoor, hash) key pair into two components, one permanent and the other ephemeral, and using the ephemeral keys to compute trapdoor hash collisions between digest values of two distinct messages. Unlike prior schemes by Chen et al. [26] and Ateniese et al. [4], the proposed scheme

ensures that it is computationally infeasible for a third party to compute an additional collision with the knowledge of a message pair whose trapdoor hash values are equal. We provide formal definitions of trapdoor hash functions, collision forgery resistance and key exposure resistance for schemes that use ephemeral keys for collision computation. We also present a detailed security analysis of the proposed discrete log-based trapdoor hashing scheme and prove its resistance to collision forgery and key exposure under the discrete log assumption. We present a performance comparison of the proposed scheme against existing key-exposure-free trapdoor hashing schemes. Our performance comparison demonstrates that the proposed scheme achieves superior performance compared to existing key-exposure-free trapdoor hashing schemes.

Chapter Organization: The rest of this chapter is organized as follows. In Section 3.2 we present formal definitions of trapdoor hash functions and their security properties, that are tailored towards schemes with multiple trapdoors. In Section 3.3 we present a novel discrete log-based trapdoor hashing scheme that is collision-resistant, key-exposure-free and prevents a third party from computing additional collisions given a message pair that have the same trapdoor hash values. We present a detailed analysis of the proposed scheme in Section 3.4 including its correctness, security and performance. Section 3.5 summarizes the chapter.

3.2 Definitions

In this section we present a definition of trapdoor hashing scheme that extends traditional definitions [74] by allowing constructions of trapdoor hash functions that can generate collisions using ephemeral keys. We also re-define notions of collision forgery resistance and key exposure resistance pertaining to trapdoor hash functions with ephemeral components.

A trapdoor hashing scheme with multiple trapdoors can be defined as follows:

Definition 3.2.1. *A trapdoor hashing scheme,* `TH`*, is a tuple* ⟨`ParGen`, `KeyGen`, `HashGen`, `TrapColGen`⟩ *whose components are defined as follows:*

`ParGen`**:** *A probabilistic polynomial-time (PPT) algorithm that takes a security parameter* λ *as input and outputs system public parameters* params*.*

`KeyGen`**:** *A PPT algorithm that takes* params *as input and outputs a (trapdoor, hash) key pair* (TK, HK)*.*

`HashGen`**:** *A PPT algorithm that takes* params*,* HK*, a message* m *and a random element* r *as inputs, and outputs the hash* $TH_{HK}(m, r)$*.*

`TrapColGen`: *A PPT algorithm that takes* params*, a pair* (TK, HK)*, a message* m*, a random element* r *and an additional message* $m' \neq m$ *as inputs, and outputs a collision parameter* c *and* HK' *such that* $TH_{HK}(m, r) = TH_{HK'}(m', c)$*. When* $HK' \neq HK$*, the public key* HK' *along with the corresponding private key* TK' *is called an ephemeral key pair.*

The function TH is said to be associated with the (long-term) hash key HK. In the past, researchers have suggested trapdoor hash functions with ephemeral components [3, 4]. However, traditional definitions for collision resistance of trapdoor hashing schemes [74] do not address trapdoor hash functions with multiple trapdoors. Moreover, prior definitions of key exposure resistance [3, 4] are tailored towards specific properties required for ensuring non-transferability of chameleon signatures. In this section, we provide more general definitions of collision resistance and key exposure resistance of a trapdoor hash function with ephemeral components.

Definition 3.2.2. *Let* TH *be a trapdoor hash function associated with hash key* HK *with corresponding trapdoor key* TK*. Given system parameters* params*, and hash keys* HK *and* HK'*, a collision forgery is defined as finding a tuple* $\langle m, r, m', c\rangle$ *such that:*

$$[m \neq m'] \wedge [TH_{HK}(m, r) = TH_{HK'}(m', c)]. \tag{3.1}$$

When $HK = HK'$*, the collision is called a* simple collision*, and when* $HK \neq HK'$*, the collision is called an* ephemeral collision*.*

Let $\mathcal{F}$ *be a PPT algorithm that takes hash keys* HK *and* HK'*, and system parameters* params *as inputs, and outputs the tuple* $\langle m, r, m', c\rangle$ *that satisfy the conditions in (3.1). Let* $Col(\langle m, r, m', c\rangle, HK, HK')$ *be a predicate denoting that (3.1) holds. We define the advantage of the PPT collision forger* $\mathcal{F}$ *as:*

$$\mathsf{Adv}_{\mathcal{F}}^{CFg}(\lambda) = Pr[Col(\mathcal{F}(\mathsf{params}, HK, HK'), HK, HK')]$$

The probability is over random choices of HK*,* HK'*, the size of the security parameter* λ *and the input of randomness for the algorithm* $\mathcal{F}$*.*

Collision Forgery Resistance: *A trapdoor hashing scheme* $\mathtt{TH} = \langle \mathtt{ParGen}, \mathtt{KeyGen}, \mathtt{HashGen}, \mathtt{TrapColGen}\rangle$ *is collision-forgery-resistant if, for all PPT algorithms* $\mathcal{F}$ *of polynomial time complexity in the security parameter* λ *that output* $\langle m, r, m', c\rangle$ *satisfying* $[Col(\mathcal{F}(\mathsf{params}, HK, HK'), HK, HK')] \wedge [HK = HK']$, $\mathsf{Adv}_{\mathcal{F}}^{CFg}(\lambda)$ *is negligible.*

Ephemeral Collision Forgery Resistance: *A trapdoor hashing scheme* $\mathtt{TH} = \langle \mathtt{ParGen}, \mathtt{KeyGen}, \mathtt{HashGen}, \mathtt{TrapColGen}\rangle$ *is ephemeral-collision-forgery-resistant if, for all PPT*

algorithms $\mathcal{F}$ of polynomial time complexity in the security parameter λ, that output $\langle m, r, m', c\rangle$ satisfying $[Col(\mathcal{F}(\mathsf{params}, HK, HK'), HK, HK')] \wedge [HK \neq HK']$, $\mathsf{Adv}_{\mathcal{F}}^{CFg}(\lambda)$ is negligible.

Another security property of trapdoor hashing scheme, captured by the notion of semantic security, is that a trapdoor hash value $h = TH_{HK}(m, r)$ must not reveal any information about the possible message m that was hashed. Let X be a random variable that takes on a finite set of values $x_1, x_2, \ldots, x_n$, with probability $P(X = x_i) = p_i$, where $0 \leq p_i \leq 1$ for each i, $1 \leq i \leq n$, and $\sum_{i=1}^{n} p_i = 1$. The *entropy* of X, denoted by $H[X]$, is the measure of information provided by an observation of X, or equivalently, the measure of uncertainty about the outcome of an experiment before an observation of X. The entropy of X is defined to be $H[X] = \sum_{i=1}^{n} p_i \lg(\frac{1}{p_i})$. Let $H[X|Y = y]$ denote the entropy of the variable X given the value of a random variable Y. Since, `HashGen` is a PPT algorithm, the value $TH_{HK}(\cdot, \cdot)$ is a random variable distributed over all strings in the range of $TH_{HK}(\cdot, \cdot)$, where the probability space is the set of all possible outcomes resulting from the random moves of `HashGen`. Let $\mathcal{M}$ denote the message space. Let m denote a random variable that takes on the finite set of values in the domain of TH from the message space $\mathcal{M}$. Semantic security of a trapdoor hashing scheme is defined as follows:

Definition 3.2.3 ([4]). *A trapdoor hashing scheme is said to be semantically security if, for all hash keys HK, the conditional entropy $H[m|h]$ of the message m given its trapdoor hash value $h = TH_{HK}(m, r)$ equals the total entropy $H[m]$ of the message space $\mathcal{M}$.*

A well-known property of trapdoor hash function is with regards to key exposure resistance [4]. Informally, a trapdoor hashing scheme is said to suffer from the *key exposure problem* if given the system parameters params and the tuple $\langle m, r, m', r', HK, HK'\rangle$ such that $TH_{HK}(m, r) = TH_{HK'}(m', r')$, the trapdoor key TK corresponding to the hash key HK can be computed in polynomial time. A trapdoor hashing scheme is said to be *resistant to key exposure* or *key-exposure-free* if, given the same inputs as before, it is computationally infeasible to compute TK. Formally, the key exposure problem and key exposure resistance can be defined as follows:

Definition 3.2.4. *Let TH be a trapdoor hash function associated with hash key HK with corresponding trapdoor key TK. Given the system parameters params, hash key HK, and the tuple $\langle m, r, m', r', HK'\rangle$ such that $TH_{HK}(m, r) = TH_{HK'}(m', r')$, key exposure is the problem of finding the trapdoor key TK corresponding to HK.*

Let $\mathcal{K}$ be a PPT algorithm that takes system parameters params*, a hash key HK, and a tuple $\langle m, r, m', r', HK' \rangle$ such that $TH_{HK}(m, r) = TH_{HK'}(m', r')$ as input and outputs the trapdoor key TK corresponding to hash key HK. Let $Col(TK, HK)$ be a predicate denoting that the trapdoor key TK corresponds to the hash key HK. We define the advantage of the PPT collision forger $\mathcal{K}$ as:*

$$\mathsf{Adv}_{\mathcal{K}}^{KExp}(\lambda) = Pr[Col(\mathcal{K}(\mathsf{params}, HK, m, r, m', r', HK'), HK)]$$

The probability is over random choices of HK, the size of the security parameter λ and the input of randomness for the algorithm $\mathcal{K}$.

Key Exposure Resistance: *A trapdoor hashing scheme* TH $= \langle$ParGen, KeyGen, HashGen, TrapColGen$\rangle$ *is key-exposure-resistant if, for all PPT algorithms $\mathcal{K}$ of time complexity polynomial in the security parameter λ, that output TK satisfying* $[Col(\mathcal{K}($params, HK, m, r, m', r', HK'), $HK)]$, $\mathsf{Adv}_{\mathcal{K}}^{KExp}(\lambda)$ *is negligible.*

3.3 Proposed key-exposure-free trapdoor hashing scheme, DL-MTH

The trapdoor hashing scheme, DL-TH, described in Section 2.3.2 suffers from the key-exposure problem. In this section, we use DL-TH as a basis for construction of an efficient, collision-resistant, key-exposure-resistant trapdoor hashing scheme, DL-MTH.

Intuitively, the trapdoor hashing scheme DL-TH suffers from key exposure because extracting the trapdoor key given a collision involves solving a simple equation with one unknown [cf. Eqn. 2.1, Section 2.3.2]. To prevent this, we utilize multiple trapdoors in our collision computation. The use of multiple trapdoors for collision computation has the effect of converting the equation for extracting the trapdoor key (given a collision) from having one unknown, into an equation with two unknowns, which in turn, makes it computationally infeasible to compute the trapdoor key given a collision producing message pair.

The ParGen, KeyGen, HashGen and TrapColGen algorithms of the proposed DL-based trapdoor hashing scheme, called DL-MTH are executed as follows:

DL-MTH.ParGen: Entities choose and agree upon common system public parameters params $= \langle p, q, \alpha, H \rangle$, where p and q are 1024-bit and 160-bit primes, respectively, $q \mid p-1$, α is an element of order q in $\mathbb{Z}_p^*$ and $H : \{0,1\}^* \mapsto \mathbb{Z}_q^*$ is a cryptographic hash function.

DL-MTH.KeyGen: An entity uses the system public parameters params, to generate its (private) trapdoor key and (public) hash key pair, $(TK, HK) = (y, Y)$, where $y \in_R \mathbb{Z}_q^*$ and $Y = \alpha^y \in \mathbb{Z}_p^*$.

`DL-MTH.HashGen:` An entity generates a trapdoor hash of a message $m \in \mathbb{Z}_q^*$ using the hash key Y, by choosing an element $r \in \mathbb{Z}_q^*$ and computing the hash as $TH_Y(m,r) = \alpha^{H(m||Y)}Y^r \mod q$. The function, TH is said to be associated with hash key Y.

`DL-MTH.TrapColGen:` Given the trapdoor key and hash key pair (TK, HK), $m, r \in \mathbb{Z}_q^*$ and an additional message $m'(\neq m) \in \mathbb{Z}_q^*$, an entity finds a trapdoor collision parameter $c \in \mathbb{Z}_q^*$ as follows:

1. Choose an ephemeral trapdoor key $z \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral hash key $Z = \alpha^z \in \mathbb{Z}_p^*$.
2. Using trapdoor keys y and z, solve for c such that $TH_Y(m,r) = TH_Z(m',c)$. Compute c as $c = z^{-1}(H(m||Y) - H(m'||Z) + yr) \mod q$.

In the schemes by Chen et al. [26] and Ateniese et al. [3, 4], an entity computes a trapdoor hash value using an ephemeral component, called a transaction identifier, i, as $h = TH_{HK}(m,r,i)$. Collisions can be computed using the long-term trapdoor key under the same transaction identifier, i.e., given the tuple $\langle m,r,i\rangle$, an entity with the trapdoor key TK can generate $\langle m',r'\rangle$ such that $TH_{HK}(m,r,i) = TH_{HK}(m',r',i)$. Due to the message hiding property of the key-exposure-resistant trapdoor hashing schemes by Chen et al. and Ateniese et al., revealing a collision $\langle m,r,,m',r',i\rangle$ allows any third party to compute $\langle m'',r''\rangle$ such that $TH_{HK}(m,r,i) = TH_{HK}(m',r',i) = TH_{HK}(m'',r'',i)$ without compromising the long-term trapdoor key. Thus, the schemes by Chen et al. [26] and Ateniese et al. [3, 4] can never be used in schemes where a signer authenticates a new message by finding collisions between the hash of the original signed-message and the new message that needs to be signed.

Unlike the schemes by Chen et al. [26] and Ateniese et al. [3, 4], the `DL-MTH` scheme allows collisions of the form $TH_{HK}(m,r) = TH_{HK'}(m',r')$, where revealing the tuple $\langle m,r,,m',r',HK'\rangle$, does not allow a third party to forge additional collisions or reveal the trapdoor key [cf. Section 3.4.2]. Thus, the form of collisions produced by the proposed scheme can be used in authentication schemes that use collision computation using the trapdoor key to authenticate new messages.

Notice that in the proposed scheme, an entity is *required* to use multiple keys (long-term and ephemeral) during collision computation. Without this requirement, i.e., if we compute a collision using only the long-term key as $c = y^{-1}(H(m||Y) - H(m'||Y) + yr) \mod q$, the resulting tuple $\langle m,r,m',c\rangle$, with $TH_Y(m,r) = TH_Y(m',c)$ can be used to compute the long-term trapdoor key. The requirement of using ephemeral keys for collision computation

in the proposed DL-MTH scheme is similar to the requirement of using ephemeral keys to compute signatures using the ElGamal family of signature schemes [38].

3.4 Analysis of the DL-MTH trapdoor hashing scheme

We present a theoretical analysis of the proposed DL-based trapdoor hashing scheme, DL-MTH, including correctness, security and performance.

3.4.1 Correctness

In this section, we demonstrate that given system parameters params, the trapdoor key and hash key pair (TK, HK), the pair $m, r \in \mathbb{Z}_q^*$ and an additional message $m'(\neq m) \in \mathbb{Z}_q^*$, the collision parameter c that is computed according to the DL-MTH.TrapColGen procedure described in Section 3.3 results in generating a collision between the trapdoor hash values of m and m'.

During computation of c, we know that $Z = \alpha^z \in \mathbb{Z}_p^*$ and $c = z^{-1}(H(m||Y) - H(m'||Z) + yr) \mod q$. Thus,

$$\begin{aligned} TH_Z(m', c) &= \alpha^{H(m'||Z)} Z^c \\ &= \alpha^{H(m'||Z)} Z^{z^{-1}(H(m||Y) - H(m'||Z) + yr)} \\ &= \alpha^{H(m'||Z)} \alpha^{(H(m||Y) - H(m'||Z) + yr)} \\ &= \alpha^{(H(m||Y) + yr)} \\ &= \alpha^{(H(m||Y)} Y^r = TH_Y(m, r) \end{aligned}$$

Thus, the technique to compute trapdoor hash collisions in the proposed DL-MTH is correct.

3.4.2 Security

In this section we provide a detailed security analysis of the proposed DL-based trapdoor hashing scheme DL-MTH. We prove the following theorems to show that the difficulty of forging collisions and key exposure in the DL-MTH scheme is based on the difficulty of solving the discrete logarithm problem in subgroup of $\mathbb{Z}_p^*$.

Theorem 3.4.1. *The proposed trapdoor hashing scheme* DL-MTH *is collision forgery resistant and ephemeral collision forgery resistant.*

Proof: We prove the forgery resistance property of the proposed trapdoor hashing scheme by showing that the discrete log problem in subgroup of $\mathbb{Z}_p^*$ reduces to collision forgery, thus violating the well known discrete log assumption.

Based on Defn. 3.2.2, we need to consider two cases: (1) When $HK = HK'$ (for collision forgery resistance); (2) When $HK \neq HK'$ (for ephemeral collision forgery resistance). The proof for the first case is well-known and the reader is referred to the proof of Theorem 2.3.1 for details.

For the case when $HK \neq HK'$, assume that there exists a PPT collision forger $\mathcal{F}$ as defined in Defn. 3.2.2 against the proposed trapdoor hashing scheme with non-negligible advantage. Given the hash keys HK, $HK'(\neq HK)$ and parameters $\langle p, q, \alpha \rangle$, $\mathcal{F}$ runs in polynomial time and outputs the tuple $\langle m, r, m', c \rangle$ such that $m \neq m'$, $r \neq c$ and $TH_{HK}(m, r) = TH_{HK'}(m', c)$ with non-negligible probability. Given $\mathcal{F}$ we can construct a PPT algorithm $\mathcal{G}$ that breaks the discrete log problem as follows. $\mathcal{G}$ is given a DLP instance $\langle p, q, \alpha, X \rangle$. $\mathcal{G}$ needs to find $x \in \mathbb{Z}_q^*$ such that $X = \alpha^x \mod p$. $\mathcal{G}$ chooses $y \in_R \mathbb{Z}_q^*$, computes $Y = \alpha^y \mod p$. $\mathcal{G}$ independently runs (in parallel) two instances of forger $\mathcal{F}$, where each instance of $\mathcal{F}$ executes with independent randomness on $\langle p, q, \alpha, X, Y \rangle$ as inputs, until each instance of $\mathcal{F}$ produces the collision forgeries $\langle m_1, r_1, m_1', c_1 \rangle$ and $\langle m_2, r_2, m_2', c_2 \rangle$, respectively. If $m_1 = m_2$ or $r_1 = r_2$ or $m_1' = m_2'$ or $c_1 = c_2$ repeat execution of $\mathcal{F}$. Given that $TH_{HK}(m_1, r_1) = TH_{HK'}(m_1', c_1)$ and $TH_{HK}(m_2, r_2) = TH_{HK'}(m_2', c_2)$, we obtain the following two linear equations:

$$H(m_1||X) + xr_1 = H(m_1'||Y) + yc_1 \mod q$$
$$H(m_2||X) + xr_2 = H(m_2'||Y) + yc_2 \mod q$$

It is straightforward to argue that the two equations are linearly independent, and thus, can be solved for x and y. This concludes the proof. □

Theorem 3.4.2. *The proposed trapdoor hashing scheme,* `DL-MTH` *is key-exposure-resistant.*

Proof: Observing the modified collision computation in `DL-MTH`, key exposure resistance implies that given two tuples $\langle m, r, HK \rangle$ and $\langle m', r', HK' \rangle$, such that $m \neq m'$, $r \neq r'$, $HK \neq HK'$ and $TH_{HK}(m, r) = TH_{HK'}(m', r')$, the probability that a PPT algorithm outputs TK is negligible.

To the contrary, assume that there exists a PPT algorithm that outputs TK in polynomial time with non-negligible probability. The discrete log of hash key HK' can be computed as:

$$TK' \equiv (r')^{-1}(H(m||HK) + TK * r - H(m'||HK')) \mod q$$

This contradicts the well known discrete log assumption. Thus, the proposed trapdoor hashing scheme is resistant to key exposure. □

In addition to collision forgery resistance and key exposure-freeness, the proposed trapdoor hashing scheme also provides semantic security. A trapdoor hashing scheme is said to be semantically secure if, for all hash keys HK, and all pairs of messages m and m', $m \neq m'$, the probability distributions of the hash values $TH_{HK}(m, r)$ and $TH_{HK}(m', r)$ are computationally indistinguishable. We omit the proof here as it closely follows the technique by Ateniese et al. [4].

3.4.3 Performance

Table 3.1 shows a performance comparison of the proposed scheme, DL-MTH with the schemes of Harn et al. (MCDLTH and MCRSATH) [37] and a scheme by Ateniese et al. (NRTH) [4] that provide features similar to the proposed scheme. In Table 3.1, the term x denotes the cost of modular exponentiation with 160-bit exponent and 1024-bit modulus, and e denotes the cost of modular exponentiation with 1024-bit exponent and 1024-bit modulus,.

Table 3.1: Performance comparison of proposed trapdoor hashing scheme DL-MTH with existing schemes.

	MCDLTH [37]	MCRSATH [37]	NRTH [4]	DL-MTH
Hash computation	$3x$	$1e$	$2x$	$2x$
Collision computation	$1x$	$1e$	$1x$	$1x$
Public key size (bits)	1024	1024	1024	1024
Hash size (bits)	160	1024	160	160
Underlying Problem	DL	RSA	DL	DL

From the table we can see that the proposed scheme DL-MTH and the NRTH schemes are more efficient than the other schemes for hash computation. The MCDLTH scheme requires an additional exponentiation compared to the proposed scheme. The RSA-based MCRSATH scheme requires exponentiations with 1024-bit exponent which tend to be approximately 6 times slower compared to exponentiations with 160-bit exponents required by the DL-based schemes. Moreover, the MCRSATH is the least efficient in terms of collision computation cost, public key size and hash size compared to all three DL-based schemes. The DL-MTH, NRTH and MCDLTH are equally efficient during collision computation, require public keys of the same size and produce the same length hash value.

Overall, the DL-MTH and the NRTH offer similar performance across all parameters, with the MCDLTH scheme being less efficient for computing the hash value, and the MCRSATH being less efficient in all respects. However, the exponentiation operation during collision computation in the NRTH scheme must be performed online. On the other hand, the proposed

scheme allows the exponentiation during collision computation to be performed offline, in which case, the online phase of collision computation in the proposed scheme is considerably faster compared to the NRTH scheme. Thus, the DL-MTH offers the best overall performance compared with existing schemes of [37] and [4].

3.5 Summary

Trapdoor hash functions were originally designed for constructing non-interactive and non-transferable signature schemes, called chameleon signatures [44]. Chameleon signatures are verifiable by no one other than the intended recipient. To provide non-repudiation and message hiding in chameleon signatures, majority of existing trapdoor hashing schemes [3, 4, 26, 44, 74] allow an entity to compute a trapdoor hash collision with a third message given two messages that result in a collision of their respective trapdoor hash values. Moreover, the handful (to be specific, only three) of existing trapdoor hash functions [4, 37] that provide the property of preventing additional collisions are computationally more expensive than conventional signatures. Thus, existing trapdoor hash functions are not suitable for use in signature schemes that use the process of computing hash collisions using a trapdoor key, rather than generating conventional signatures using a secret key, to authenticate messages.

In this chapter, we presented a key-exposure and collision-resistant, discrete log-based trapdoor hash function that is efficient to compute and ensures that it is computationally infeasible for a third party to compute additional hash collisions with the knowledge of a message pair whose trapdoor hash values are equal. We provided formal definitions of trapdoor hash functions and their security for schemes that use ephemeral keys for collision computation. We also presented a theoretical analysis, including correctness, security and performance, of the proposed trapdoor hashing scheme. In our security analysis we proved the resistance to collision forgery and key exposure under the discrete log assumption. Our performance comparison of the proposed scheme against existing key-exposure-free trapdoor hashing schemes demonstrates that the proposed scheme achieves superior performance in terms of computation overhead during trapdoor hash and collision computation compared to existing schemes.

Chapter 4

Efficient proxy signatures based on trapdoor hash functions

4.1 Introduction

Today, many large-scale and complex distributed systems like peer-to-peer networks, pervasive and ubiquitous computing environments, Cloud systems, and Grids are highly dynamic, decentralized and loosely coupled with constituent components that are spread throughout a network. Researchers have advocated agent-based computing as the natural computational model for such systems [40]. For instance, agent-based computing has been used in Grids, by employing agents (also called proxies) to eliminate the need to have the user online during long-lived computations, scheduling access to resources and map computations onto resources [17, 31]. Industry, government, and academia have been developing applications for mobile agents for use in telecommunications systems, personal digital assistants, information management, and e-commerce [62] where clients need to search for special services or products, negotiate with potential business entities and perform remote operations on behalf of some other clients. More recently, Weissman et al. [81] proposed the use of agents in Cloud computing systems to accelerate applications by performing optimized operations at strategic network locations in the Cloud with high bandwidth to/from a Cloud service relative to the service user.

4.1.1 Addressing security in agent-based systems

Use of agent-based computing in multi-Cloud, Grid and other distributed systems opens up new avenues for adversarial users to compromise the system. Security in Clouds and other IT systems heavily rely on establishing trust relationships among the actors involved. In agent-based computing, users have to establish trust relationships with agents which includes their security, reliability, availability, and business continuity guarantees. Moreover, sensitive information stored and processed by agents need to be protected from exposure, alteration and corruption.

One technique of establishing trust relationships with agents is by strengthening the role of Service Level Agreements (SLAs) to include security guarantees on top of QoS. However, in the context of securing Agent-based systems, SLAs are currently in their nascent form

and require considerable advancements to mature. In the meantime, researchers need to focus on developing mechanisms that would aid in providing these security guarantees. These mechanisms include encryption, authentication, authorization and integrity checks.

Agents-based security solutions have been used in Grids [17, 31] to provide authentication and access control. Combination of user agents and resource agents in Grids have been used for implementing security policies that can support single sign-on, interoperability among different local policies, and authenticated and authorized access to resources in the Grid [17, 31]. Agent-optimized multi-Cloud applications can also benefit from a Grid-like security architecture, where we can enhance the capabilities of agents to perform authentication and authorization on behalf of a service user in addition to accelerating Cloud applications. An effective technique for providing authentication (in addition to authorization) in agent-based systems is by use of proxy signatures.

4.1.2 Application of proxy signatures in agent-based systems

Proxy delegation is a process by which an entity, the delegator, transfers its signing rights and capabilities to another entity, the proxy. Following delegation, the proxy can generate signatures on behalf of the delegator. The process of proxy delegation has been classified into three broad categories, namely, full delegation, partial delegation and delegation-by-warrant or certificate. Full delegation is a rather intuitive solution in which the delegator securely transfers its secret key to the proxy. This requires absolute trust on the proxy, a secure channel and provides unrestricted signing rights to proxy. In a partial delegation scheme, the delegator also uses a secure channel to transfer a delegation key derived from its secret key to the proxy. The proxy then derives its proxy signing key pair from the delegation key. However, the proxy still maintains unrestricted signing capabilities. As opposed to full and partial delegation that require secure channel and absolute trust on the proxy, delegation-by-warrant approach eliminates these impractical requirements [13]. A warrant embodies a set of business and security policies agreed in advance by the delegator and the proxy that restrict the signing capabilities of the proxy. The delegator generates a warrant and a signature on the warrant, called a certificate, and sends the (warrant, certificate) pair to the proxy. The proxy generates signatures — using its own private key — on messages that must conform to the warrant, and includes the (warrant, certificate) pair in the resulting signatures. Any entity wanting to verify a proxy signature must check the validity of the proxy signature as well as delegator's agreement on the signed message.

In agent-based systems, users would play the role of delegators and delegate their signing rights to agents (proxies) using warrants that limit signing capabilities of agents. Agents could span multiple administrative domains with different local security policies in each domain. In such scenarios, warrants can map security policies between different domains and allow integrating local policies of multiple domains into the global security framework. A proxy signature not only provides assurance that an agent is the authentic originator of a message, but also that the agent is legitimately acting on behalf of a user within constraints specified in the warrant. Moreover, proxy signatures simplify security auditing and obtaining support for investigations by providing non-repudiation.

4.1.3 Problem statement

The importance of proxy signatures has been repeatedly highlighted by applied cryptographers: variations of proxy signatures include threshold proxy signatures [78], blind proxy signatures [6], proxy signatures with warrant recovery [45] and so forth. However, many recent developments in the area of proxy signatures rely on heuristic arguments demonstrating security rather than providing provable security guarantees [13]. Moreover, formal construction of provably secure proxy signatures by Boldyreva et al. [13], Malkin et al. [56] and Schuldt et al. [73] have introduced unnecessary complexity in the design of proxy signatures which lead to high computational and communication overhead.

4.1.4 Contributions

In this chapter, we make the following contributions towards securing multi-agent systems:

1. We propose an elegant technique to construct a provably secure proxy signature using trapdoor hash functions [3, 4, 44, 74] that can be used to authenticate and authorize agents acting on behalf of users in agent-based computing systems. The generic technique allows the choice of primitives open to policy specifications. Moreover, security of the proposed technique does not rely on availability of a secure channel.

2. The proposed generic technique only requires execution of a conventional signature scheme during the offline *delegation* phase and not during the online *proxy signature generation* phase; this improvement is due to the use of trapdoor hash functions [59, 74] as a building block. Thus, the proposed technique inherently provides the efficiency of online/offline signature schemes [29].

3. We demonstrate the effectiveness of our approach for creating practical instances by applying properties of discrete log-based cryptosystem to produce an efficient proxy

signature scheme. We compare the performance of the discrete log-based scheme with existing schemes. Our performance analysis shows a clear improvement in the performance over prior schemes, which can be attributed to our proxy signature construction technique.

Chapter Organization: The rest of this chapter is organized as follows. We present a generic technique to construct proxy signature schemes using trapdoor hash functions and provide formal definitions and security specifications in Section 4.2 and provide a detailed proof of security of the generic construction using the random oracle model [11]. In Section 4.3, we use primitives from discrete log based cryptosystem to present an efficient instantiation of the proposed technique. We perform a theoretical analysis of the DL-based proxy signature scheme in Section 4.4. We summarize the Chapter in Section 4.5.

4.2 Technique to construct proxy signatures using trapdoor hash functions

In this section, we present the basic idea behind the construction of proxy signature schemes using trapdoor hash functions followed by formal definitions, security specifications and semantics of trapdoor hash-based proxy signature schemes. Finally we present the proposed generic trapdoor hash-based proxy signature scheme.

4.2.1 Basic idea

We begin with providing the basic idea behind the construction of proxy signature schemes using trapdoor hash functions. We also provide heuristic arguments in support of informal security requirements like undeniability, identifiability and verifiability that were introduced by Lee et al. [47]. A formal security treatment of the proposed scheme is provided later in the chapter.

Fig. 4.1 shows the basic idea behind the construction of a trapdoor hash-based proxy signature scheme. A proxy signature scheme is divided into four phases, namely, *initialization, proxy delegation, proxy signature generation* and *proxy signature verification.* During the initialization phase, all entities choose and agree upon common system public parameters and generate their (public, private) key pairs. An entity (the delegator) wanting to delegate signing rights to another entity (the proxy), does the following:

1. Creates an appropriate warrant. The delegator and the proxy agree upon warrant specifications by out of band mechanisms, perhaps using and abiding by formal business policies. The structure of the warrant plays an important role in restricting

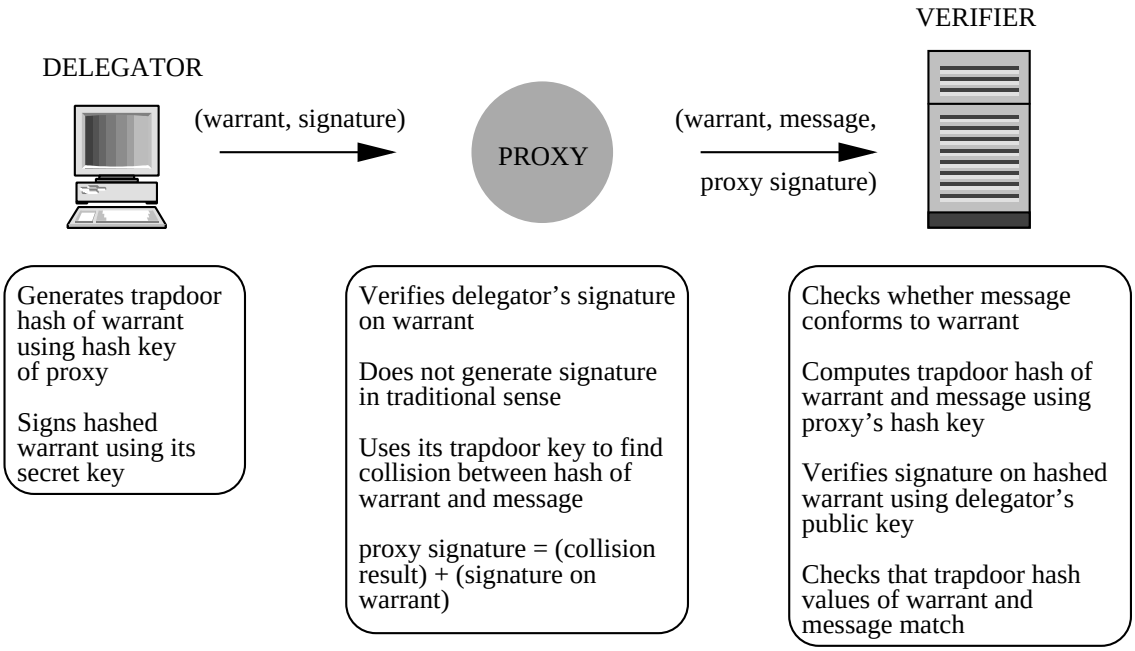

Figure 4.1: Basic technique of using a trapdoor hash function to generate proxy signatures.

the proxy's signing rights by dictating the types of messages that the proxy can sign, the validity period and other restrictions [13, 73].

2. Computes a trapdoor hash of the warrant. The associated hash key belongs to the proxy being designated.

3. Generates a signature on the trapdoor hash of warrant. In the instantiation of the proposed generic proxy signature scheme, we use the provably secure Schnorr variant [72]. However, the choice of the signing equation is open to policy specifications. The delegator's signature on the warrant *prevents misuse of signing rights* by illegitimate modifications of the warrant by a malicious delegator, proxy, or any other adversary.

4. Sends the (warrant, signature) pair to the proxy over an insecure channel. We assume the channel is reliable and do not consider network reliability issues.

After receiving the (warrant, signature) pair, the proxy checks the warrant specifications. If the warrant is bogus, the proxy aborts; otherwise, it verifies the delegator's signature on the warrant. This completes the proxy delegation phase.

Note, unlike most common proxy signature schemes [13, 47, 43], the proxy signing key pair is *not* derived from the delegator's signature, in the proxy delegation phase. In our

scheme, the proxy does not generate a signature on a message, in the traditional sense, on behalf of the delegator. Instead, the proxy uses its trapdoor key, known exclusively to itself, to find a trapdoor collision between the trapdoor hash of the warrant and the given message. This guarantees *undeniability*. The proxy then tags the result of the collision along with the delegator's signature, warrant and message to collectively generate the proxy signature.

Any entity that wants to verify the proxy signature does the following:

1. Checks whether the message conforms to the warrant.
2. Obtains the hash key of the proxy from a publicly available directory and computes the trapdoor hash of the warrant. This provides *strong identifiability*.
3. Checks the delegation agreement by verifying the delegator's signature on the hashed warrant. If verification fails, the verifier aborts. This guarantees *verifiability*.
4. Computes the trapdoor hash of the message using the result of the collision contained in the proxy signature.
5. Verifies that the proxy indeed stamped the message with the secret trapdoor key, by comparing the trapdoor hash of the message and the warrant. If the hash values do not match, the verifier aborts.

Verification of subsequent proxy signatures, exchanged by the same (proxy, verifier) pair gets more efficient: the verifier computes the trapdoor hash of the message and compares it with the previously computed trapdoor hash of warrant. Intuitively, this improvement in efficiency happens because the delegator's signature on the warrant contained within the proxy signature does not change from one message to another.

4.2.2 Definitions, security specifications and semantics

We now formally define a trapdoor hash-based proxy signature scheme. We assume existence of a public key infrastructure, where each entity is associated with a certificate signed by a certificate authority (CA). This public key certificate binds the entity's identity with its public key. The CA is responsible for ensuring that no two identities are bound to the same public key (i.e., public keys are unique). The CA is also responsible for differentiating keys used for signing and trapdoor hashing and including the purpose of key usage in the certificate. To further simplify our constructions, we assume that identities are unique to each entity and cannot be spoofed (for eg., by generating identities as hash of public keys). Thus, users can be uniquely identified by their public keys, and vice versa, using

certificates, and without relying on proofs of knowledge [73]. Moreover, we also assume a publicly available directory, from which entities can retrieve public key certificates of other entities, given their identities. We explicitly include warrants and its structure as part of our definition and security model to avoid trivial attacks involving malicious alteration of warrants [13].

Definition 4.2.1. *A trapdoor hash-based proxy signature scheme* TPS *is the tuple* ⟨ParGen, KeyGen, SigGen, SigVer, (Delegate, Accept), PSigGen, PSigVer⟩ *whose components are defined as follows:*

ParGen: *A PPT algorithm that takes a security parameter λ as input and outputs system public parameters* params.

KeyGen: *A PPT algorithm that takes* params *as input and outputs a (private, public) key pair (SK, PK) and a (trapdoor, hash) key pair (TK, HK).*

SigGen: *A PPT algorithm that takes* params*, a message m and SK as inputs and outputs a signature σ on m.*

SigVer: *A deterministic algorithm that takes* params*, PK, m and σ as inputs and outputs* Valid *if σ was generated on m using SK and* Invalid *otherwise.*

Delegate, Accept: *A pair of interactive algorithms for proxy delegation that are defined as follows:*

Delegate: *A PPT algorithm that takes* params*, SK and HK as inputs and computes a delegation certificate* cert *containing the following: (1) A warrant w that contains a message space descriptor as defined in [13] and identities of delegator and proxy; (2) A random element r; (3) A signature σ generated using SK on $TH_{HK}(w, r)$.* Delegate *has no local output.* Delegate *will interact with* Accept *for delegation of signing rights.*

Accept: *A deterministic algorithm that takes* params, cert, *PK and HK as input, and outputs* $\langle \text{Accept}, \text{cert} \rangle$ *if: (1) w conforms to agreement between delegator and proxy, and (2) σ was generated on $TH_{HK}(w, r)$ using SK; and* $\langle \text{Reject}, \perp \rangle$ *otherwise.*

PSigGen: *A PPT proxy signature generation algorithm that takes* params, cert*, a message m and TK as inputs and outputs a proxy signature σ_P on m conforming to warrant w.*

`PSigVer`: *A deterministic algorithm that takes* params, PK, HK, m *and* σ_P *as inputs and outputs* Valid *if: (1)* m *conforms to* w*, and (2)* σ_P *was generated on* m *using* TK*; and* Invalid *otherwise.*

Modeling the Adversary. Our security model is based on models by Boldyreva et al. [13] and Schuldt et al. [73]. Our security model integrates components of both models, along with some modifications specifically tailored toward trapdoor hash-based proxy signature schemes. Similar to the models by Boldyreva et al. and Schuldt et al., our model does not require the adversary to have knowledge of the private key of dishonest users playing the security game, i.e., our model is not in the registered key model. Thus, our model captures attacks where the adversary registers public keys for which the corresponding private key is not known [73]. Also, similar to prior models, our underlying security game involves an adversary that has the ability to play the role of any arbitrary entity except a single honest entity. The adversary attempts to forge a regular or a proxy signature of the single honest entity. The adversary is given access to standard and proxy signing oracle. The adversary can interact with the honest entity multiple times playing the role of different entities each time.

The issue of proxy signing key exposure differentiates our security model from models by Boldyreva et al. and Schuldt et al. Schuldt et al. were the first to model the possibility of proxy signing key exposure in proxy signature schemes — the adversary is given the capability to gain knowledge of proxy signing keys, while playing the security game with an honest entity, which can be potentially used to forge standard or proxy signatures. Schuldt et al. argue that proxy signatures are often used in applications where signing is performed in hostile environments. While secure storage is available for a proxy's long-term secret key, the proxy signing key, stored in a less trusted device, is vulnerable to exposure/compromise. Thus, secure proxy signatures should not leak information about the long-term key of the proxy when the proxy signing key is exposed.

While in Schuldt et al.'s model, any arbitrary proxy signing key of the honest entity can be exposed to the adversary, Boldyreva et al.'s model only allows exposure of proxy signing keys that are generated during self-delegation by the honest entity. The different treatment of proxy key exposure arises because, the technique to avoid leakage of information about the long-term key in presence of proxy signing key exposure requires a fresh proxy key pair to be generated during delegation which, in case of Boldyreva et al., only occurs during self delegation, whereas, in case of Schuldt et al., occurs during every delegation run.

Unlike the schemes by Boldyreva et al., Schuldt et al. and other common proxy signature schemes [43, 47, 56], the proposed trapdoor hash-based proxy signature scheme does not involve generation of any proxy signing key. In the proposed scheme, the proxy finds collisions between trapdoor hashes of the warrant and the given message, and tags the result of the collision along with the delegation certificate to collectively generate a proxy signature. Collision computation involves use of the long-term trapdoor key. Thus, all signature generation operations in the proposed scheme involve use of long-term keys[1]. This excludes our scheme from application in environments where delegation is performed to protect long-term keys. Fortunately, existing applications of proxy signatures like securing mobile agent communication, Grid computing, and multi-Cloud systems, are not intended for protection of long-term keys, but rather for providing authentication (in addition to authorization) in absence of the original signer in a delegation-based framework. Thus, the possibility of exposure of proxy signing keys is excluded from our security model. However, our scheme can be modified to be proven secure in the models allowing key exposure. Informally, similar to the approaches by Boldyreva et al. and Schuldt et al., we can also necessitate the generation of a fresh (trapdoor, hash) key pair during each delegation run. In this case, the long-term private (signing) key of the proxy would only be used to certify the per-delegation (trapdoor, hash) key pair. As long as a secure signature scheme is used in the per-delegation certification process, we can allow exposure of per-delegation private (trapdoor) key without any leakage of information regarding the long-term key.

Security Model. We now formally describe our security model. Let E be a set of entities containing a single honest entity e. The remaining entities $e' \in E - \{e\}$ are corrupted by an adversary $\mathcal{A}$. The goal of the adversary $\mathcal{A}$, modeled as a PPT algorithm, is to output one or more of the following forgeries:

Type I: $\mathcal{A}$ outputs a forged signature of e on a message m', where e never signed message m'.

Type II: $\mathcal{A}$ outputs a forged proxy signature of e on message m' conforming to warrant w' on behalf of an arbitrary entity e', where one of the following holds: (1) e' never delegated e; (2) e' delegated e under warrant $w \neq w'$; (3) e' delegated e under warrant w' but e did not generate a proxy signature on m' under the warrant w'.

[1]We note that the long-term key used for proxy signature generation is *not* a signing key that is used for conventional signature generation. Instead, the proxy uses its trapdoor key for proxy signature generation. The trapdoor key is used exclusively for collision computation and the long-term *signing* key of proxy is never used in our scheme.

Type III: $\mathcal{A}$ outputs a forged proxy signature of an arbitrary entity e' on a message m' conforming to warrant w' on behalf of entity e where one of the following holds: (1) e never delegated e' as a proxy; (2) e delegated e' under a different warrant $w \neq w'$.

The adversary is allowed to select and register public keys and hash keys for any corrupted entity e', play the role of e' as a delegator while performing proxy delegation or request e to participate in a proxy delegation playing the role of a proxy. There is no limitation on the number of times the adversary can request any two entities in E to perform proxy delegation. The adversary controls all communications, in the sense that no secure channel exists between a proxy and a delegator. However, we assume that the channel is reliable with usual connotations of message delivery. Next, we describe the game that adversary $\mathcal{A}$ plays with entity e in an attempt to forge a standard or proxy signature.

Specifications of the Game. Consider a game $G_{\mathcal{A}}^{TPS}(\lambda)$, where λ is a security parameter. The game is played by adversary $\mathcal{A}$ with entity e in an attempt by $\mathcal{A}$ to produce a Type I, Type II or a Type III forgery.

The game $G_{\mathcal{A}}^{TPS}(\lambda)$ is initialized as follows. All entities in E choose and agree upon the system public parameters params. Entity e generates its (private, public) key pair (SK_e, PK_e) and its (trapdoor, hash) key pair (TK_e, HK_e), and registers (PK_e, HK_e) in a publicly available directory. The adversary $\mathcal{A}$ is given access to PK_e, HK_e, a signing oracle $\mathcal{O}_S$ and a proxy signature generating oracle $\mathcal{O}_{PS}$ that are described below. The adversary creates four empty sets M_s, M_p, E' and D_{ee}. In addition, the adversary creates empty sets $D_{ee'}$ and $D_{e'e}$ for each $e' \in E - \{e\}$. At any point during the game the adversary $\mathcal{A}$ can generate (or derive) the (public, hash) key pair $(PK_{e'} \neq PK_e, HK_{e'} \neq HK_e)$ of an entity $e' \in E - \{\{e\} \cup E'\}$ and register $(PK_{e'}, HK_{e'})$ in a publicly available directory. After each key registration for entity e', E' is set to $E' \cup \{e'\}$. The game is executed as follows:

The adversary $\mathcal{A}$ makes the following requests to the honest entity e in any order and any number of times.

- Request e to delegate $e' \in E'$: $\mathcal{A}$ can request e to delegate $\mathcal{A}$ as a proxy, playing the role of e'. Entity e executes $\mathtt{Delegate}(\mathsf{params}, SK_e, HK_{e'}) \rightarrow (cert_{ee'} = \langle w_{ee'}, r_{ee'}, \sigma_e \rangle)$ and sends $cert_{ee'}$ to $\mathcal{A}$. If $\mathtt{Accept}(\mathsf{params}, cert_{ee'}, PK_e, HK_{e'})$ outputs *Accept* then $D_{ee'}$ is set to $D_{ee'} \cup \{(w_{ee'}, r_{ee'}, \sigma_e)\}$, otherwise $\mathcal{A}$ aborts.

- Request $e' \in E'$ to delegate e: $\mathcal{A}$ can request e to be delegated as a proxy on behalf of $\mathcal{A}$ playing the role of e'. $\mathcal{A}$ executes $\mathtt{Delegate}(\mathsf{params}, SK_{e'}, HK_e) \rightarrow (cert_{e'e} =$

$\langle w_{e'e}, r_{e'e}, \sigma_{e'}\rangle$) and sends $cert_{e'e}$ to e. If $\texttt{Accept}(\mathsf{params}, cert_{e'e}, PK_{e'}, HK_e)$ outputs *Accept*, then $D_{e'e}$ is set to $D_{e'e} \cup \{(w_{e'e}, r_{e'e}, \sigma_{e'})\}$, otherwise e aborts.

- Request e to delegate itself: $\mathcal{A}$ can request e to delegate itself. Entity e executes $\texttt{Delegate}\ (\mathsf{params}, SK_e, HK_e) \rightarrow (cert_{ee} = \langle w_{ee}, r_{ee}, \sigma_e\rangle)$ and sends $cert_{ee}$ to $\mathcal{A}$. If $\texttt{Accept}(\mathsf{params},\ cert_{ee},\ PK_e,\ HK_e)$ outputs *Accept*, then D_{ee} is set to $D_{ee} \cup \{(w_{ee}, r_{ee}, \sigma_e)\}$.

The adversary $\mathcal{A}$ makes the following queries to oracles $\mathcal{O}_S$ and $\mathcal{O}_{PS}$ in any order and any number of times:

- Query oracle $\mathcal{O}_S$: $\mathcal{A}$ queries oracle $\mathcal{O}_S$ with message m and obtains a signature σ on m valid under PK_e. M_s is set to $M_s \cup \{m\}$.

- Query oracle $\mathcal{O}_{PS}$: $\mathcal{A}$ queries oracle $\mathcal{O}_{PS}$ with message m, warrant w_{xe}, $l \in \mathbb{N}$ and $x \in E' \cup \{e\}$ as input. If $D_{xe}[l]$ is not defined or $(w_{xe}, \cdot, \cdot) \notin D_{xe}[l]$, where $D_{xe}[l]$ is l-th element in the set D_{xe}, the query is invalid and oracle returns $\perp$. Otherwise, the oracle returns $\sigma_P \leftarrow \texttt{PSigGen}(\mathsf{params}, D_{xe}[l], m, TK_e)$ and M_p is set to $M_p \cup \{(w_{xe}, m)\}$.

Outcome of the Game. At the completion of the game $G_{\mathcal{A}}^{TPS}(\lambda)$, the adversary outputs a forgery of the form σ' or σ'_P. The outcome of the game is decided as follows:

1. If $\mathcal{A}$ outputs a Type I forgery, σ' on message m', which implies $[Valid \leftarrow \texttt{SigVer}\ (\mathsf{params},\ PK_e,\ m',\ \sigma')] \wedge [m' \notin M_s]$, then $G_{\mathcal{A}}^{TPS}(\lambda)$ outputs *Success*.

2. If $\mathcal{A}$ outputs a Type II forgery, σ'_P on message m' conforming to warrant w' under the delegation parameter $cert' = \langle w', r', \sigma'\rangle$, which implies $[\exists x \in E' \cup \{e\}] \wedge [valid \leftarrow \texttt{PSigVer}(\mathsf{params}, \sigma'_P,\ PK_x,\ HK_e)] \wedge [(w', m') \notin M_p]$, then $G_{\mathcal{A}}^{TPS}(\lambda)$ outputs *Success*.

3. If $\mathcal{A}$ outputs a Type III forgery, σ'_P on message m' conforming to warrant w' under the delegation parameter $cert' = \langle w', r', \sigma'\rangle$ which implies $[\exists x \in E'] \wedge [valid \leftarrow \texttt{PSigVer}(\mathsf{params}, \sigma'_P,\ PK_e,\ HK_x)] \wedge [(w', r', \sigma') \notin D_{ex}]$, then $G_{\mathcal{A}}^{TPS}(\lambda)$ outputs *Success*.

4. Else, $G_{\mathcal{A}}^{TPS}(\lambda)$ outputs *Failure*.

Adversary $\mathcal{A}$, on input (PK_e, HK_e) and security parameter λ, plays the game $G_{\mathcal{A}}^{TPS}(\lambda)$ with entity e. Define the advantage of $\mathcal{A}$ as: $Adv_{\mathcal{A}}^{TPS}(\lambda) = Pr[Success \leftarrow G_{\mathcal{A}}^{TPS}(\lambda)]$. The trapdoor hash-based proxy signature scheme, TPS is *secure against adaptive chosen message attack* if $Adv_{\mathcal{A}}^{TPS}(\lambda)$ is negligible for all PPT algorithms $\mathcal{A}$ of time complexity polynomial in the security parameter λ.

4.2.3 Trapdoor hash-based proxy signatures schemes, TPS

Let an entity D act as the delegator, wanting to delegate signing rights to another entity P, the proxy. Let TH be a trapdoor hash function that computes key exposure-free collisions.

A naïve implementation of the basic idea presented in Section 4.2.1 poses some weaknesses. During delegation, a delegator generates a signature, σ on the trapdoor hash of a warrant, $TH_{HK}(w, r)$ using the hash key of a proxy. The proxy can use its trapdoor key, TK to find $\langle w', c, HK' \rangle$ such that $TH_{HK}(w, r) = TH_{HK'}(w', c)$, essentially forging the delegator's signature on a new warrant, w' of its choice. To prevent this simple forgery, we require the delegator to generate a signature on $TH_{HK}(w, r)||w$ instead of $TH_{HK}(w, r)$ during proxy delegation. Now, even though the proxy can find a collision between w and w' such that $TH_{HK}(w, r) = TH_{HK'}(w', c)$, we observe that $TH_{HK}(w, r)||w \neq TH_{HK'}(w', c)||w'$, thus, preventing the forgery.

With the aforementioned fix in place, after a proxy obtains the delegator's signature σ on $m = TH_{HK}(w, r)||w$, the proxy can still produce a forged standard signature of the delegator by outputting σ on $m' = TH_{HK'}(w', c)||w$ (existential forgery), where $w' \neq w$ and $TH_{HK}(w, r) = TH_{HK'}(w', c)$. To avoid this, we introduce a mechanism to differentiate between delegator's standard signatures and signatures created for delegation, by requiring that standard signatures be generated on $1||m$ instead of m, similar to the technique introduced by Boldyreva et al. [13].

We now present a generic construction of trapdoor hash-based proxy signature scheme, TPS. The ParGen and KeyGen algorithms are as defined in Definition 4.2.1. The system public parameters are given by params. The delegator D's long-term (private, public) key pair is given by (SK, PK). The proxy P's long-term (trapdoor, hash) key pair is given by (TK, HK). The remaining components of the proposed trapdoor hash-based proxy signature scheme, TPS are defined as follows:

TPS.SigGen: Given system parameters params, and a message m, the delegator performs the standard signing operation of the underlying signature scheme on $1||m$ using its secret key, SK and outputs the signature σ.

TPS.SigVer: Given system parameters params, a message m, public key PK and a candidate signature σ, an entity prepends 1 to m and verifies σ on $1||m$ under public key PK following the verification algorithm of the underlying signature scheme. If verification is successful the verifier outputs $Valid$, otherwise outputs $Invalid$.

TPS.Delegate: The delegator D delegates a proxy P as follows:

1. Chooses a random element r and generates a warrant w containing identities D, P and a message space descriptor.
2. Computes trapdoor hash of warrant as $TH_{HK}(w, r)$, where HK is the hash key of P, and generates a signature σ on $h = TH_{HK}(w, r)||w$ using its private key SK.
3. Sends the delegation certificate $cert = \langle \sigma, w, r \rangle$ to P.

`TPS.Accept:` After receiving the delegation parameters from D, the proxy P performs the following operations:

1. Checks whether warrant w conforms to the agreement with D. If check fails, outputs $\langle Reject, \perp \rangle$ and aborts.
2. Computes $TH_{HK}(w, r)$ and sets $h = TH_{HK}(w, r)||w$.
3. Verifies signature σ on h under public key PK. If verification is successful outputs $\langle Accept, cert \rangle$, otherwise, outputs $\langle Reject, \perp \rangle$.

`TPS.PSigGen:` Given a message m, and delegation certificate $cert$, P generates a proxy signature σ_P as follows:

1. Checks whether the message m conforms to restrictions specified in the warrant w. If check fails, aborts.
2. Computes collision parameter c, and HK' using TK such that $TH_{HK}(w, r) = TH_{HK'}(m, c)$, where TK is the trapdoor key of P.
3. Outputs the proxy signature $\sigma_P = \langle cert, m, c, HK' \rangle$ on the message m conforming to warrant w.

`TPS.PSigVer:` Given a proxy signature σ_P, an entity can verify the delegation agreement, identify the proxy and verify the proxy signature on message m conforming to warrant w as follows:

1. Checks whether the message m conforms to warrant specifications w. If check fails, outputs *Invalid* and aborts.
2. Looks up the hash key HK of P and public key PK of D from a publicly available directory.
3. Computes $TH_{HK}(w, r)$, sets $h = TH_{HK}(w, r)||w$ and verifies signature, σ on h under public key PK. If verification fails, outputs *Invalid* and aborts.

4. Otherwise, computes $TH_{HK'}(m, c)$ and checks whether $TH_{HK'}(m, c) = TH_{HK}(w, r)$. If check fails, outputs *Invalid*, otherwise, outputs *Valid*.

To verify P's subsequent proxy signature, $\sigma_P = \langle cert, m'', c'', HK'' \rangle$ on message m'', an entity only computes $TH_{HK''}(m'', c'')$ and checks whether the hash value equals h. If the hash values are equal, the proxy signature is valid. Verification of σ on $h||w$ is not repeated.

The security of the proposed proxy signature scheme can be summarized by the following theorem.

Theorem 4.2.1. *The proposed proxy signature scheme,* TPS*, is secure provided that the underlying standard signature scheme is secure (as defined in [13]), and TH is semantically secure, key-exposure-resistant, collision-forgery-resistant and ephemeral-collision-forgery-resistant.*

Proof: We show that for every adversary $\mathcal{A}$ against the trapdoor hash-based proxy signature scheme, TPS, with non-negligible advantage $Adv_{\mathcal{A}}^{TPS}(\lambda)$, we can construct an adversary $\mathcal{B}$, which breaks the underlying signature scheme, or forges a trapdoor collision with non-negligible advantage. All entities agree upon system parameters params. Entity e generates its (private, public) key pair (SK_e, PK_e) and the (trapdoor, hash) key pair (TK_e, HK_e). The adversary $\mathcal{B}$ is given access to a signing oracle $\mathcal{O}_{S_e}$ that generates standard signatures using SK_e by performing the signing operation of the underlying signature scheme and a trapdoor collision oracle $\mathcal{O}_{CL_e}$ that generates collisions for the trapdoor hash function associated with HK_e (i.e., $\mathcal{O}_{CL_e}$ takes messages m, m', and random element r as inputs and outputs $\langle c, HK' \rangle$ such that $TH_{HK_e}(m, r) = TH_{HK'}(m', c)$).

Adversary $\mathcal{B}$ runs $\mathcal{A}$ (both modelled as PPT algorithms) on inputs PK_e and HK_e and answers requests and queries made by $\mathcal{A}$ as follows:

- When adversary $\mathcal{A}$ requests $\mathcal{B}$ to register $PK_{e'} \neq PK_e$ and $HK_{e'} \neq HK_e$, $\mathcal{B}$ stores $PK_{e'}$ and $HK_{e'}$ ($SK_{e'}$ and $TK_{e'}$ are unknown to $\mathcal{B}$). $\mathcal{B}$ initializes two empty sets $D_{ee'}$ and $D_{e'e}$. E' is set to $E' \cup \{e'\}$.

- When $\mathcal{A}$ requests e to designate $\mathcal{A}$ as a proxy, playing the role of $e' \in E'$, $\mathcal{B}$ does the following: (1) Creates a warrant $w_{ee'}$ and chooses $r_{ee'}$; (2) Computes $m = TH_{HK_{e'}}(w_{ee'}, r_{ee'e})||w_{ee'}$ and queries oracle $\mathcal{O}_{S_e}$ with m as input and obtains a signature σ_e (using SK_e) on m; (3) Sends $cert_{ee'} = \langle w_{ee'}, r_{ee'}, \sigma_e \rangle$ to $\mathcal{A}$. $\mathcal{A}$ checks whether σ_e is a valid signature on m under PK_e. If σ_e is valid, then $D_{ee'}$ is set to $D_{ee'} \cup \{cert_{ee'}\}$, otherwise $\mathcal{B}$ aborts.

- When $\mathcal{A}$ requests e to be designated as a proxy on behalf of $\mathcal{A}$ playing the role of $e' \in E'$, $\mathcal{A}$ does the following: (1) Creates a warrant $w_{e'e}$ and chooses $r_{e'e}$; (2) Generates a signature $\sigma_{e'}$ on $m = TH_{HK_e}(w_{e'e}, r_{e'e})||w_{e'e}$; (3) Sends $cert_{e'e} = \langle w_{e'e}, r_{e'e}, \sigma_{e'} \rangle$ to $\mathcal{B}$. $\mathcal{B}$ checks whether $\sigma_{e'}$ is a valid signature on m under $PK_{e'}$. If $\sigma_{e'}$ is valid, then $D_{e'e}$ is set to $D_{e'e} \cup \{cert_{e'e}\}$, otherwise, $\mathcal{B}$ aborts.

- When $\mathcal{A}$ requests e to designate itself, $\mathcal{B}$ does the following: (1) Creates a warrant w_{ee} and chooses r_{ee}; (2) Computes $m = TH_{HK_e}(w_{ee}, r_{ee})||w_{ee}$ and queries oracle $\mathcal{O}_{S_e}$ with m as input and obtains a signature σ_e (using SK_e) on m; $\mathcal{B}$ sends $cert_{e'e} = \langle w_{ee}, r_{ee}, \sigma_e \rangle$ to $\mathcal{A}$. $\mathcal{A}$ checks whether σ_e is a valid signature on m under PK_e. If σ_e is valid, then D_{ee} is set to $D_{ee} \cup \{cert_{ee}\}$, otherwise $\mathcal{B}$ aborts.

- When $\mathcal{A}$ queries oracle $\mathcal{O}_S$ with message m, $\mathcal{B}$ makes the query $1||m$ to its signing oracle $\mathcal{O}_{S_e}$ and forwards the response σ_e to $\mathcal{A}$. M_s is set to $M_s \cup \{m\}$.

- When $\mathcal{A}$ queries oracle $\mathcal{O}_{PS}$ with message m, warrant w_{xe}, $l \in \mathbb{N}$ and $x \in E' \cup \{e\}$ as input, $\mathcal{B}$ responds as follows: If $D_{xe}[l]$ is not defined or $(w_{xe}, \cdot, \cdot) \notin D_{xe}[l]$, the query is invalid and $\mathcal{B}$ returns $\perp$. Otherwise $\mathcal{B}$ parses $D_{xe}[l]$ as $(w_{xe}, r_{xe}, \sigma_x)$. $\mathcal{B}$ queries oracle $\mathcal{O}_{CL_e}$ with inputs m, w_{xe}, r_{xe} to obtain a response $\langle c, HK' \rangle$. $\mathcal{B}$ returns the proxy signature $\sigma_P = \langle cert_{xe}, m, c, HK' \rangle$ to $\mathcal{A}$. M_p is set to $M_p \cup \{(w_{xe}, m)\}$.

The adversary outputs a forgery of the form σ' or σ'_P. If $\mathcal{A}$'s forgery is a Type I forgery of the form σ' on message m', the signature σ' is in fact a forged standard signature on message $1||m'$ valid under public key PK_e, such that $1||m' \notin M_s$. $\mathcal{B}$ outputs the forgery σ' on message $1||m'$ valid under PK_e.

If $\mathcal{A}$'s forgery is a Type II forgery of the form σ'_P on message m' conforming to warrant w', $\mathcal{B}$ parses σ'_P as $\langle cert', m', c', HK' \rangle$. Next, $\mathcal{B}$ parses $cert'$ as $\langle w', r', \sigma' \rangle$, where σ' is a signature on $TH_{HK_e}(w', r')$ valid under PK_x ($x \in E' \cup \{e\}$) and $(w', m') \notin M_p$. $\mathcal{B}$ outputs a collision forgery $\langle w', r', m', c', HK' \rangle$ such that $TH_{HK_e}(w', r') = TH_{HK'}(m', c')$, under the hash key HK_e.

If $\mathcal{A}$'s forgery is a Type III forgery of the form σ'_P on message m' conforming to warrant w', $\mathcal{B}$ parses σ'_P as $\langle cert', m', c', HK' \rangle$. Next, $\mathcal{B}$ parses $cert'$ as $\langle w', r', \sigma' \rangle$, where σ' is a signature on $TH_{HK}(w', r')||w'$ ($x \in E'$) valid under PK_e and $cert' \notin D_{ex}$. $\mathcal{B}$ outputs the forgery σ' on message $m' = TH_{HK}(w', r')||w'$ valid under public key PK_e. □

4.3 A discrete log-based proxy signature scheme, DL-TPS

Performance sensitive settings, such as mobile agent applications [47, 46], require efficient proxy signature. In this chapter, we demonstrate the potential of our generic approach [cf. Section 4.2.3] to create practical instances by using cryptographic primitives from discrete log-based cryptosystem to construct an efficient trapdoor hash-based proxy signature scheme.

Let an entity D act as the delegator, wanting to delegate signing rights to another entity P, the proxy. The `ParGen`, `KeyGen`, `Delegate`, `Accept`, `PSigGen` and `PSigVer` algorithms of trapdoor hash-based proxy signature scheme, `DL-TPS`, are executed as follows:

DL-TPS.ParGen: Entities choose and agree upon common system public parameters $\mathsf{params} = \langle p, q, \alpha, H, G \rangle$, where p and q are 1024-bit and 160-bit primes, respectively, $q \mid p-1$, α is an element of order q in $\mathbb{Z}_p^*$ and $H, G : \{0,1\}^* \mapsto \mathbb{Z}_q^*$ are cryptographic hash functions.

DL-TPS.KeyGen: The delegator D chooses its long-term private key $x \in_R \mathbb{Z}_q^*$ and computes the corresponding long-term public key as $X = \alpha^x \in \mathbb{Z}_p^*$. The proxy P chooses its long-term trapdoor key $y \in_R \mathbb{Z}_q^*$ and computes the corresponding long-term hash key as $Y = \alpha^y \in \mathbb{Z}_p^*$.

DL-TPS.SigGen: Given system parameters params, a message m the delegator D generates a standard signature on m as follows:

1. Choose an ephemeral private key $k \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral public key $r = \alpha^k \in \mathbb{Z}_p^*$.
2. Solve for t in the equation: $t \equiv k + xH(m||r) \mod q$ (this is the well-known `DL-Schnorr` [72] signature scheme). Signature on m is given by $\sigma = \langle t, r \rangle$.

DL-TPS.SigVer: Given a candidate signature σ on message m, system parameters params and public key X, verification proceeds as follows:

1. If $r = \alpha^t X^{-H(m||r)} \mod q$, the signature $\sigma = \langle t, r \rangle$ on message m is valid under the public key X. Output $Valid$.
2. Otherwise, output $Invalid$ and abort.

DL-TPS.Delegate: The delegator D delegates a proxy P as follows:

1. Choose an ephemeral private key $k \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral public key $r = \alpha^k \in \mathbb{Z}_p^*$.
2. Generate a warrant w and compute the trapdoor hash of w as $TH_Y(w, r) = \alpha^{h_w} Y^r \mod q$, where $h_w = H(w||Y)$. The warrant w specifies restrictions on the messages the proxy signer is allowed to sign.
3. Solve for t in the equation: $t \equiv k + xG(TH_Y(w, r)||w||r) \mod q$ (Schnorr [72]-type signing).
4. Form the delegation certificate *cert* containing the signature, $\sigma = \langle t, r \rangle$ and warrant, w, and send *cert* to P.

`DL-TPS.Accept:` After receiving the delegation certificate $cert = \langle \sigma, w \rangle$, where $\sigma = \langle t, r \rangle$, from D, the proxy P performs the following operations:

1. Check whether warrant w conforms to the agreement with D. If check fails, output $\langle Reject, \perp \rangle$ and abort.
2. Compute $TH_Y(w, r) = \alpha^{h_w + yr} \mod q$ and $h = G(TH_Y(w, r)||w||r)$.
3. If $r = \alpha^t X^{-h} \mod q$, the signature $\sigma = \langle t, r \rangle$ on h is valid under the public key X of D. Output $\langle Accept, cert \rangle$. Otherwise, output $\langle Reject, \perp \rangle$.

`DL-TPS.PSigGen:` Given a message m, P generates a proxy signature σ_P as follows:

1. Check whether the message m conforms to restrictions specified in the warrant w. If check fails, abort.
2. Choose an ephemeral trapdoor key $z \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral hash key $Z = \alpha^z \in \mathbb{Z}_p^*$.
3. Compute $h_m = H(m||Z)$ and using trapdoor key y and ephemeral key z, solve for c in the equation: $c = z^{-1}(h_w - h_m + yr) \mod q$.
4. The proxy signature σ_P is the tuple $\langle cert, m, c, h_m \rangle$ on the message m conforming to warrant w. Send σ_P to the verifier V.

`DL-TPS.PSigVer:` Given a proxy signature σ_P, V can verify the delegation agreement, identify the proxy and verify the proxy signature on message m conforming to warrant w as follows:

1. Check whether the message m conforms to warrant w. If check fails, output *Invalid* and abort.

2. Look up the hash key Y of P from a publicly available directory. Compute $TH_Y(w,r) = \alpha^{h_w} Y^r \mod q$ and $h = G(TH_Y(w,r)||w||r)$.
3. Compute $r' = \alpha^t X^{-h} \mod q$. If $r' \neq r \mod q$, output *Invalid* and abort.
4. Compute $Z' = \alpha^{(h_w - h_m)c^{-1}} Y^{rc^{-1}} \mod p$. Check whether $h_m = H(m||Z')$. If check fails, output *Invalid*. Otherwise, output *Valid*.

To verify P's subsequent proxy signatures on a new message m, the verifier V only computes $Z' = \alpha^{(h_w - h_m)c^{-1}} Y^{rc^{-1}} \mod p$ and checks whether $h_m = H(m||Z')$. If the hash values are equal, the proxy signature is valid. Next, we perform a detailed analysis of the proposed proxy signature scheme, `DL-TPS`.

4.4 Analysis of the `DL-TPS` proxy signature scheme

We present a theoretical analysis of the proposed DL-based proxy signature scheme, `DL-TPS`, including correctness, security and performance.

4.4.1 Correctness

A proxy signature scheme constructed following the procedures in Section 4.2 is correct if an arbitrary proxy signature, $\sigma_P = \langle cert, m, c, h_m \rangle$ on message m conforming to warrant w, generated by a proxy P on behalf of D, passes the proxy signature verification procedure `DL-TPS.PSigVer`, using the long term public key of D and the long-term hash key of P, provided: (1) All entities choose and agree upon the system public parameters $\mathsf{params} = \langle p, q, \alpha, H, G \rangle$; (2) D and P honestly execute key generation algorithm `DL-TPS.KeyGen`; (3) D honestly executes proxy delegation signature generation algorithm, `DL-TPS.Delegate` scheme, and (4) P honestly uses his long-term trapdoor key y and ephemeral trapdoor key z to compute c such that $TH_Y(w,r) = TH_Z(m,c)$.

Proposition 4.4.1. *The proposed DL-based proxy signature scheme,* `DL-TPS` *is correct.*

Proof: Provided the aforementioned correctness conditions hold, the proxy signature $\sigma_P = \langle cert, m, c, h_m \rangle$ on message m conforming to warrant w passes the proxy signature verification procedure `DL-TPS.PSigVer`, using the long-term public key of D and the long-term hash key of P if the following conditions are met: (1) D's signature σ on $h = G(TH_Y(w,r)||w||r) \mod q$ passes verification following the signature verification algorithm of `DL-TPS.Accept` and (2) The hash value, $TH_Y(w,r) \mod q$ equals $TH_Z(m,c) \mod q$.

We first show that the signature verification algorithm of `DL-TPS.Accept` executed on the signature $\sigma = \langle t, r \rangle$ on h, under the public key X outputs $\langle cert, Accept \rangle$. During verification

of σ we observe:

$$\alpha^t X^{-h} = \alpha^{(k+xh-hx)} = \alpha^k = r$$

Thus, the signature σ on h is valid under public key of D.

Next we show that the value of $c \in \mathbb{Z}_q^*$ computed during execution of `DL-TPS.PSigGen` represents a collision between $TH_Y(w, r)$ and $TH_Z(m, c)$. This can be seen as follows:

$$TH_Z(m, c) = \alpha^{h_m + zc} = \alpha^{h_m + z(z^{-1}(h_w - h_m + yr))} = \alpha^{h_w + yr} = TH_Y(w, r)$$

What remains to be shown is that the quantity Z' computed during execution of `DL-TPS.PSigVer` equals the ephemeral hash key Z computed by the proxy during execution of `DL-TPS.PSigGen`. This can be seen as follows:

$$\alpha^{(h_w - h_m)c^{-1}} Y^{rc^{-1}} = \alpha^{c^{-1}(h_w - h_m + yr)} = \alpha^{z(h_w - h_m + yr)^{-1}(h_w - h_m + yr)} = \alpha^z = Z$$

where, $c = z^{-1}(h_w - h_m + yr) \mod q$. □

4.4.2 Security

In this section we provide a detailed security analysis of the the proposed DL-based proxy signature scheme `DL-TPS`, following notions of security against adaptive chosen message attack.

Given that `DL-MTH` exhibits properties of collision forgery resistance, ephemeral collision forgery resistance, key exposure resistance and semantic security, and the provable security guarantees provided by the well-known `DL-Schnorr` [72] signature scheme, we state and prove the following theorem.

Theorem 4.4.2. *The DL-based proxy signature scheme* `DL-TPS` *is secure against adaptive chosen message attack in the random oracle model [11] under the DL assumption.*

Proof: We show that for every adversary $\mathcal{A}$ against the DL-based proxy signature scheme with non-negligible advantage, $Adv_{\mathcal{A}}^{DL-TPS}(\lambda)$, we can construct an adversary $\mathcal{B}$, which solves the DL problem with non-negligible advantage $Adv_{\mathcal{B}}^{DL}(\lambda)$. The hash functions H and G behave as a random oracles (denoted as $\mathcal{O}_H$ and $\mathcal{O}_G$, respectively). The system parameter param is given as the tuple $\langle p, q, \alpha, H, G \rangle$. Entity e generates its (private, public) key pair (SK_e, PK_e) and the (trapdoor, hash) key pair (TK_e, HK_e). The adversary $\mathcal{B}$ is given access to the public keys PK_e and HK_e, and simulates signing oracle, $\mathcal{O}_S$, the proxy signature generating oracle, $\mathcal{O}_{PS}$ and the hashing oracles, $\mathcal{O}_H$ and $\mathcal{O}_G$.

Adversary $\mathcal{B}$ runs $\mathcal{A}$ (both modelled as PPT algorithms) on inputs PK_e and HK_e and answers requests and queries made by $\mathcal{A}$ as follows:

- When adversary $\mathcal{A}$ requests $\mathcal{B}$ to register $PK_{e'} \neq PK_e$ and $HK_{e'} \neq HK_e$, $\mathcal{B}$ stores $PK_{e'}$ and $HK_{e'}$ ($SK_{e'}$ and $TK_{e'}$ are unknown to $\mathcal{B}$). $\mathcal{B}$ initializes two empty sets $D_{ee'}$ and $D_{e'e}$. E' is set to $E' \cup \{e'\}$.

- When $\mathcal{A}$ requests e to designate $\mathcal{A}$ as a proxy, playing the role of $e' \in E'$, $\mathcal{B}$ does the following: (1) Creates a warrant $w_{ee'}$; (2) Chooses $h, t \in_R \mathbb{Z}_q^*$ and computes $r = \alpha^t PK_e^{-h} \mod p$; (3) If $H(w_{ee'}||HK_{e'})$ is defined, then retrieves $h_{w_{ee'}} = H(w_{ee'}||HK_{e'})$, otherwise chooses $h_{w_{ee'}} \in_R \mathbb{Z}_q^*$, sets $H(w_{ee'}||HK_{e'}) = h_{w_{ee'}}$ and stores $h_{w_{ee'}}$ in the *hash entry* for $H(w_{ee'}||HK_{e'})$; (4) Computes $TH_{HK_{e'}}(w_{ee'}, r)$; (5) If $G(TH_{HK_{e'}}(w_{ee'}, r)||w_{ee'}||r)$ is defined, then $\mathcal{B}$ aborts. Otherwise sets $G(TH_{HK_{e'}}(w_{ee'}, r)||w_{ee'}||r) = h$ and stores h in its *hash entry*; (6) Sets $\sigma_e = \langle t, r\rangle$ and sends $cert_{ee'} = \langle w_{ee'}, \sigma_e\rangle$ to $\mathcal{A}$. $\mathcal{A}$ checks whether σ_e is a valid signature on the hash, h (obtained by querying $\mathcal{O}_G$ with inputs $w_{ee'}$, r and $HK_{e'}$) under PK_e. If σ_e is valid, then $D_{ee'}$ is set to $D_{ee'} \cup \{cert_{ee'}\}$, otherwise $\mathcal{B}$ aborts.

- When $\mathcal{A}$ requests e to be designated as a proxy on behalf of $\mathcal{A}$ playing the role of $e' \in E'$, $\mathcal{A}$ does the following: (1) Creates a warrant $w_{e'e}$; (2) Chooses $k \in_R \mathbb{Z}_q^*$ and computes α^k; (3) Queries oracle $\mathcal{O}_G$ with $w_{e'e}$, r and HK_e as inputs to obtain $h = G(TH_{HK_e}(w_{e'e}, r)||w_{e'e}||r)$, and (4) Generates a signature $\sigma_{e'} = \langle t, r\rangle$ (using $SK_{e'}$) on h. $\mathcal{A}$ sends $cert_{e'e} = \langle w_{e'e}, \sigma_{e'}\rangle$ to $\mathcal{B}$. $\mathcal{B}$ checks whether $\sigma_{e'}$ is a valid signature on h under PK_e. If $\sigma_{e'}$ is valid, then $D_{e'e}$ is set to $D_{e'e} \cup \{cert_{e'e}\}$, otherwise, $\mathcal{B}$ aborts.

- When $\mathcal{A}$ requests e to designate itself, $\mathcal{B}$ does the following: (1) Creates a warrant w_{ee}; (2) Chooses $h, t \in_R \mathbb{Z}_q^*$ and computes $r = \alpha^t PK_e^{-h} \mod p$; (3) If $H(w_{ee})$ is defined, then retrieves $h_{w_{ee}} = H(w_{ee}||HK_e)$, otherwise chooses $h_{w_{ee}} \in_R \mathbb{Z}_q^*$, sets $H(w_{ee}||HK_e) = h_{w_{ee}}$ and stores $h_{w_{ee}}$ in the *hash entry* for $H(w_{ee}||HK_e)$; (4) Computes $TH_{HK_e}(w_{ee}, r)$; (5) If $G(TH_{HK_e}(w_{ee}, r)||w_{ee}||r)$ is defined, then aborts, otherwise sets $G(TH_{HK_e}(w_{ee}, r)||w_{ee}||r) = h$ and stores h in its *hash entry*; (6) Sets $\sigma_e = \langle t, r\rangle$ and sends $cert_{ee} = \langle w_{ee}, \sigma_e\rangle$ to $\mathcal{A}$. $\mathcal{A}$ checks whether σ_e is a valid signature on hash of w_{ee} (obtained by querying $\mathcal{O}_G$ with inputs w_{ee},r and HK_e) under PK_e. If σ_e is valid, then D_{ee} is set to $D_{ee} \cup \{cert_{ee}\}$, otherwise $\mathcal{B}$ aborts.

- When $\mathcal{A}$ queries oracle $\mathcal{O}_S$ with message m, $\mathcal{B}$ does the following: (1) Chooses $t, h \in_R \mathbb{Z}_q^*$ and computes $r = \alpha^t PK_e^{-h} \mod p$; (2) Checks whether $H(m||r)$ is defined, and if so, aborts; (3) Otherwise, sets $H(m||r) = h$ and $\sigma_e = \langle t, r\rangle$. $\mathcal{B}$ sends σ_e to $\mathcal{A}$. M_s is set to $M_s \cup \{m\}$.

- When $\mathcal{A}$ queries oracle $\mathcal{O}_{PS}$ with message m, warrant w_{xe}, $l \in \mathbb{N}$ and $x \in E' \cup \{e\}$ as input, $\mathcal{B}$ responds as follows: If $D_{xe}[l]$ is not defined or $(w_{xe}, \cdot) \notin D_{xe}[l]$, the query is invalid and $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ parses $D_{xe}[l]$ as $(w_{xe}, \sigma_x = \langle t, r \rangle)$, and performs the following operations: (1) Chooses $c \in_R \mathbb{Z}_q^*$; (2) Retrieves h and $h_{w_{xe}}$, where $h = G(TH_{HK_e}(w_{xe}, r)||w_{xe}||r)$ and $h_{w_{xe}} = H(w_{xe}||HK_e)$; (3) Chooses $h_m \in_R \mathbb{Z}_q^*$ and computes $Z = \alpha^{(h_{w_{xe}} - h_m)c^{-1}} HK_e^{rc^{-1}} \mod p$; (4) If $H(m||Z)$ is defined then aborts, otherwise sets $H(m||Z) = h_m$ and stores h_m in the hash entry for $H(m||Z)$. $\mathcal{B}$ sends $\sigma_P = \langle w_{xe}, \sigma_x, m, c, h_m \rangle$, to $\mathcal{A}$. M_p is set to $M_p \cup \{(w_{xe}, m)\}$.

- When $\mathcal{A}$ queries oracle $\mathcal{O}_G$ with message m, an element r and hash key HK_x ($x \in E' \cup \{e\}$) as inputs, $\mathcal{B}$ returns h if $G(TH_{HK_x}(m, r)||m||r)$ is defined, i.e., $\exists h$ such that $h = G(TH_{HK_x}(m, r)||m||r)$. Otherwise, $\mathcal{B}$ does the following: (1) If $H(m||HK_x)$ is defined, retrieves $h_m = H(m||HK_x)$, otherwise, chooses $h_m \in_R \mathbb{Z}_q^*$, sets $H(m||HK_x) = h_m$ and stores h_m in the *hash entry* for $H(m||HK_x)$; (2) Uses HK_x, r, h_m to compute $TH_{HK_x}(m, r)$; (3) Chooses $h \in_R \mathbb{Z}_q^*$, sets $G(TH_{HK_x}(m, r)||m||r) = h$, stores h in the *hash entry* for $G(TH_{HK_x}(m, r)||m||r)$ and returns h to $\mathcal{A}$.

- When $\mathcal{A}$ queries oracle $\mathcal{O}_H$ with message m and element r as inputs, $\mathcal{B}$ returns h if $H(m||r)$ is defined, that is if $\exists h$ such that $h = H(m||r)$. Otherwise, $\mathcal{B}$ chooses $h \in_R \mathbb{Z}_q^*$, sets $H(m||r) = h$, stores h as the *hash entry* for $H(m||r)$ and returns h to $\mathcal{A}$.

- When $\mathcal{A}$ queries oracle $\mathcal{O}_H$ with message m as input, $\mathcal{B}$ returns h if $H(m)$ is defined, that is if $\exists h$ such that $h = H(m)$. Otherwise, $\mathcal{B}$ chooses $h \in_R \mathbb{Z}_q^*$, sets $H(m) = h$, stores h as the *hash entry* for $H(m)$ and returns h to $\mathcal{A}$.

The adversary outputs a forgery of the form σ' or σ'_P. If $\mathcal{A}$'s forgery is a Type I forgery of the form σ' on message m', $\mathcal{B}$ uses the (provably secure) forged Schnorr signature [72] to break the discrete log problem and output SK_e.

If $\mathcal{A}$'s forgery is a Type II forgery of the form σ'_P on message m' conforming to warrant w', $\mathcal{B}$ parses σ'_P as $\langle \sigma', c', h'_m \rangle$ and σ' as $\langle r', t' \rangle$. $\mathcal{B}$ computes $h_{w'} = H(w'||HK_e)$ and $Z' = \alpha^{(h_{w'} - h'_m)c'^{-1}} HK_e^{r'c'^{-1}} \mod p$. Since, $(w', m') \notin M_p$ we know that $\langle w', r', m', c', Z' \rangle$ represents a collision forgery, i.e., $TH_{HK_e}(w', r') = TH_{Z'}(m', c')$, where Z' is computed as in `DL-TPS.PSigVer`. Following Theorem 3.4.1, $\mathcal{B}$ uses the forged collision to break the discrete log problem and output TK_e corresponding to HK_e.

If $\mathcal{A}$'s forgery is a Type III forgery of the form σ'_P on message m' conforming to warrant w', $\mathcal{B}$ parses σ'_P as $\langle \sigma', c', h'_m \rangle$ and σ' as $\langle r', t' \rangle$. In this case, e is the delegator and $e' \in E'$

is the proxy. If $\mathcal{A}$ did not make the query $(w', r', HK_{e'})$, where $r' = \alpha^{k'} \mod q$, to oracle $\mathcal{O}_G$, then $\mathcal{B}$ aborts. Otherwise, let h' be the response $\mathcal{B}$ gave when $\mathcal{A}$ made this query. $\mathcal{B}$ rewinds $\mathcal{A}$ to the point where $\mathcal{A}$ makes the $(w', r', HK_{e'})$ query to $\mathcal{O}_G$ and gives $\mathcal{A}$ a new randomly chosen value $h'' \neq h' \in_R \mathbb{Z}_q^*$. $\mathcal{B}$ continues the execution of $\mathcal{A}$, until $\mathcal{A}$ outputs a forgery σ_P'' on message m' conforming to warrant w'. If the forgery is not of the form $\langle \sigma'', \cdot, \cdot \rangle$, where $\sigma'' = \langle r', t'' \rangle$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ computes the private key SK_e corresponding to PK_e of the honest entity as: $SK_e = (t' - t'')(h' - h'')^{-1}$. Thus, $\mathcal{B}$ is able to solve the discrete logarithm problem with non-negligible probability. □

4.4.3 Performance

Table 4.1: Performance comparison of proposed proxy signature scheme DL-TPS with existing schemes.

	MUO [57]	KPW [43]	PH [69]	MLKK) [48]	HSMW [39]	ZNS [83]	DL-TPS
Delegation	$3e$	$3e$	$4e$	$4e$	$3s + 2p$	$2s + 2p$	$6e$
Proxy Sig Gen	$1e$	$1e$	$1e$	$1e$	$5s$	$2s$	$1e$
Proxy Sig Ver	$4e$	$3e$	$3e$	$4e$	$5p$	$1s + 2p$	$5e(2e)$
Public key size (bits)	2048				1532	1532	2048
Proxy Sig size (bits)	1344			1504	480	160	640
Secure Channel	Y	N	N	N	N	N	N
Provably Secure	N	Y	N	N	Y	N	Y

Table 4.1 shows a comparison of the proposed proxy signature scheme, DL-TPS, with those developed by Mambo et al. (MUO) [57], Kim et al. (KPW) [43], Petersen et al. (PH) [69], Lee et al. (MLKK) [48], Huang et al. (HSMW) [39] and Zhang et al. (ZNS) [83]. In Table 4.1, the term e denotes the cost of modular exponentiation with a 160-bit exponent, s denotes the cost of scalar multiplications and p denotes the cost of pairing computation. All computation, storage and communication costs are based on a security benchmark of 1024-bits. The size of proxy signature shown in the table excludes the size of warrant and message for all schemes. For the DL-TPS scheme, the cost of verifying initial and subsequent proxy signatures exchanged between any pair of proxy and verifier are different. The table entry for proxy signature verification cost for the DL-TPS scheme shows the cost of verifying subsequent proxy signatures exchanged between any pair of proxy and verifier in parenthesis (and the cost of initial proxy signature verification outside parenthesis).

The performance characteristics of the KPW scheme is the same as its provably secure variant by Boldyreva et al. [13] and its proxy non-designated variant by Lee et al. [47]. The scheme by Lee et al. [47], however, requires a secure communication channel between the

delegator and the proxy. Lee et al. (MLKK) [48] proposed a variant of the scheme in [47] to overcome this weakness. For the sake of uniformity in comparison, we consider a security benchmark of 1024 bits — the system public parameters of DL-TPS, MUO, KPW, MLKK, and PH are given by the tuple $\mathsf{params} = \langle p, q, \alpha \rangle$, where p and q are 1024-bit and 160-bit primes, respectively, and α is an element of order q in $\mathbb{Z}_p^*$. Also, we assume the employment of Schnorr signature for proxy signature generation in the MUO, KPW, PH and MLKK schemes.

We observe that the proposed DL-TPS scheme achieves the best proxy signature verification efficiency for subsequent messages exchanged between any pair of proxy and verifier, while maintaining comparable signature generation performance. Our proxy signature verification is 33% more efficient compared to the next most efficient proxy signature schemes, KPW and PH. Although the delegation process is expensive in DL-TPS, we argue that this step would not be performed often and the majority of computational overhead would be caused by proxy signature generations and verifications. The HSMW and ZNS schemes use considerably more expensive bilinear pairing operations in the delegation and proxy signature verification phases. To the best of our knowledge, the best known result for computing a single Tate pairing equals approximately 11110 multiplications in $\mathbb{Z}_q$, where q is a 171-bit prime (for security benchmark of 1024-bits) [8]. Moreover, the HSMW requires the largest amount of space ($10KB$) for storage of system parameters [52].

For schemes, DL-TPS, MUO, KPW, MLKK and PH, the size of the long-term public key, excluding shared components (primes p and q) equals 2048-bits. The pairing-based schemes, HSMW and ZNS, use public keys of size 1532-bits.

The proposed DL-TPS scheme also produces the smallest proxy signatures compared to MUO, KPW, PH and MLKK. Even though the HSMW and ZNS schemes produce smaller signatures, they suffer from significant computational overhead. Thus, the DL-TPS scheme achieves the best all-round performance compared with other schemes in the literature while being provably secure.

4.5 Summary

We presented a simple and elegant technique to construct a provably secure proxy signature using trapdoor hash functions that can be used to authenticate and authorize agents acting on behalf of users in agent-based computing systems. Unlike most common proxy signature schemes [13, 47, 43], the proxy signing key pair is not derived from the delegator's signature during the proxy delegation phase. Instead of generating a signature on the message in the traditional sense, the proxy uses its trapdoor key (known exclusively to itself) to find

a trapdoor collision between the trapdoor hash of the warrant and the given message. We provided a rigorous security analysis of the proposed generic construction `TPS`.

We instantiated our scheme using primitives from discrete log-based cryptosystem to yield the `DL-TPS` proxy signature scheme. We provided definitions, security specifications and performed a theoretical analysis of the proposed scheme, `DL-TPS`, including correctness, security and performance. As shown in Table 4.1, `DL-TPS` yields superior performance in terms of proxy signature verification compared to the schemes in [13, 39, 43, 47, 57, 69, 83] and produces the smallest proxy signatures compared to schemes in [13, 43, 47, 48, 57, 69]. We believe that our work plays a crucial role in providing authorization and authentication of agents in Multi-Agent Systems. In the future we plan to further investigate improvements in efficiency of our proposed technique through the use of primitives from LFSR-based cryptosystems [24, 35].

Chapter 5

Efficient stream authentication using trapdoor hash functions

5.1 Introduction

Many Web-based services involve distribution of content like digital audio, video, software, games, stock quotes, streaming presentations and live news feeds through distributed networking technologies such as content distribution networks (CDNs), multicast networks and peer-to-peer networks. Unfortunately, these modern distributed systems, designed to distribute content to a large group of users, also provide an ideal platform for adversarial users to launch a myriad of attacks with widespread consequences. Adversaries can masquerade as legitimate content providers to distribute malicious content possibly infected with worms, viruses, etc. Adversaries can also place themselves in the content distribution path, for example, by compromising Web caches [53], and modify the content in ways that can potentially harm client devices. Authenticating the content plays a crucial role in preventing these attacks.

Although CDNs like Akamai employ mechanisms for providing physical security, host system security, access control, software reliability and integrity, and 24x7 monitoring and response, these mechanisms are primarily designed for providing security of the CDN's service network infrastructure and ensure proper functioning of its distributed network of servers rather than protecting content distributed through the CDN [2]. The task of protecting content is the responsibility of content providers. Unfortunately, much of the content provided over the Internet today is transmitted without any protection mechanism. For instance Amazon, YouTube and many other highly popular content providers distribute their content over an insecure connection. Although encryption is not necessary in the instances like browsing products sold by Amazon and videos on YouTube, authenticating content is becoming increasingly necessary. Popular content providers often provide highly personalized user experience by inserting targeted advertisements and dynamically generated content. Today the majority of mass-viewed content is dynamically generated and multimedia-rich that include combination of text, audio, still images, animation, video, and interactive forms that are gathered from a myriad of sources, assembled and presented to the user. In such scenarios, malicious modification of content by malicious sources becomes

a legitimate threat. To highlight this point, recently, Google and MSN (Microsoft(R)) were observed to be distributing malware after attackers were able to masquerading as a legitimate advertising provider and inserting malicious advertisements (by exploiting two Internet Explorer, one Java, and four Adobe Reader flaws) that installed the HDD Plus malware [16]. YouTube was also a victim of an attack where malicious code was inserted into pages (by exploiting a cross-site scripting vulnerability) displaying the targeted videos that would launch when users opened the video clip redirecting users to pornographic sites and display falsified news alerts [36]. While flaws in software that were exploited were eventually patched, these attacks could be prevented by using authentication mechanisms to protect the content.

Using conventional techniques for message authentication require the sender and the receiver to have the ability to store the entire message before processing the message. However, in most instances of distributing content, like digitized multimedia, the content provider transmits the content in the form of digital streams that a receiver is required to consume at more or less the stream arrival rate without excessive delay. To protect such delay-sensitive digital streams against malicious attacks, security mechanisms need to be designed to process long sequence of bits in an efficient manner that allows the receiver to verify the authenticity of the stream in portions (to avoid possessing the entire stream before verification) without excessive processing delays associated with each portion of the stream. This is typically done by dividing the stream into blocks (or chunks) and using an efficient security mechanism to secure each block of data. In this chapter, we focus on the problem of efficient stream authentication using digital signatures. The goal is to provide integrity, origin authentication, and non-repudiation for individual blocks that comprise a digital stream.

5.1.1 Problem statement

Efficient authentication of stream poses several challenges:

1. Authentication of delay-sensitive streams requires high verification rates which translates to requiring minimal computational overhead to verify individual blocks and avoiding excessive accumulation of data in buffers before verification can proceed. For instance, to maintain jitter-free playback of on-demand media distributed through CDN's, per-block verification rates at client devices must equal or exceed the rate at which blocks arrive at the device.

2. For real-time generated digital streams, a sender must be able to sign a block as soon as it is generated with minimal computational overhead. For instance, to prevent delays in distribution of real-time content, like stock quotes and live news feeds, that can influence critical business decisions, per-block signing rates at content originators must exceed the rate at which blocks are generated.

3. Stream transmission is typically done using unreliable transport protocols, like UDP, to provide a high throughput, which can cause loss of datagrams during transmission. Thus, stream authentication mechanisms must be designed to tolerate arbitrary loss of data blocks without affecting the ability of a receiver to verify remaining received blocks.

4. Authenticating information such as signatures and hash values placed within a block (which we call per-block communication overhead) must be limited to a small, constant size to prevent excessive bandwidth utilization while transmitting signed streams.

5.1.2 Contributions

We propose a novel trapdoor hash-based signature amortization technique for efficient stream authentication. From a high-level functional perspective, the proposed technique is the first in the literature to simultaneously provide the following features:

1. The proposed technique tolerates out-of-order arrival of blocks in the stream and is resilient to transmission losses of an arbitrary number of intermediate blocks, without affecting the verifiability of remaining blocks in the stream. These properties stem from the ability of the receiver to verify each block based on the contents of *any* previously received block.

2. Our technique minimizes delays in transmitting a stream following the block-signing process and playback of the stream following the block-verification process. We do this by designing highly efficient block signing and verification procedures that require constant number of operations and limited memory.

3. The proposed technique limits communication overhead incurred while sending the authenticating material to a small, constant-sized signature that is augmented to each block in the stream. Moreover, we avoid wastage of bandwidth by not needing redundant placement of the same authenticating information in multiple blocks.

The proposed technique represents the first known attempt in the literature to use trapdoor hash functions for building a signature amortization scheme. Trapdoor hash functions provide the unique capability to find collisions between hashes of different messages using a secret trapdoor key. We use this capability to design a signature amortization technique where we compute a signature on the trapdoor hash of a block and amortize it over multiple blocks by finding trapdoor collisions with the hash of the signed block. The receiver verifies the authenticity of a block by computing its trapdoor hash value and comparing it with the hash value of any other block in the stream. The proposed technique ensures authenticity of the stream as no other entity besides the sender can generate the collisions without knowledge of the secret key. We demonstrate the efficacy of the proposed technique to build a practical stream authentication scheme by constructing a discrete-log based instantiation of the proposed technique. The resulting stream authentication scheme, called `DL-SA`, provides adaptive block verification where the receiver can choose to verify individual blocks, or batch-verify multiple blocks using less computation at the cost of maintaining a small buffer. We analyze the security of `DL-SA` and compare its performance against existing stream authentication schemes with comparable features. Our security analysis proves that `DL-SA` is resistant against forgery attacks under the discrete log assumption. Our performance comparison demonstrates that `DL-SA` incurs the least per-block communication and signature generation overheads compared to the existing schemes.

Chapter Organization: The rest of this Chapter is organized as follows. We present details of using stream authentication mechanisms in CDN's in Section 5.2. In Section 5.3, we present a novel trapdoor hash-based signature amortization technique for stream authentication and discuss security and performance merits of the proposed technique. In Section 5.4 we present a discrete-log based instantiation of the proposed technique. We perform a security and performance analysis of discrete-log based scheme, `DL-SA`, in Section 5.5. Section 5.6 summarizes the chapter.

5.2 Securing content distribution: a motivating application

In this section, we present an application of using signature amortization for stream authentication in securing on-demand and real-time content typically distributed via CDN's. We highlight specific challenges associated to efficiently authenticating content and later, present a viable approach toward solving existing problems. This application is not meant to be the sole motivation for the signature amortization mechanism we are introducing, but only an illustration of some of its potential uses and benefits. The proposed signature

amortization technique can also be applied to provide authentication in multicast and peer-to-peer networks.

5.2.1 System architecture overview

Figure 5.1 shows a high-level architecture of a CDN. The components include a core data center, multiple edge caches each serving multiple clients, and the CDN backbone (Internet or WAN). Edge caches are strategically distributed worldwide to lower content delivery costs to requesting clients. Both core data centers and edge caches consist of a media server and a content distribution manager (CDM). The media servers provide storage of media content. The media server at the core data center serves as the content origin for both on-demand and real-time content. Clients can include devices like PCs, laptops, PDAs, cell phones. A CDM provides several functionalities including: (1) Usage tracking service that enables logging, accounting and billing of content usage. (2) Caching service that implements content caching techniques. (3) Content processing service that fetches the requested content from the media server (or stores content into the media server), splits the media file into blocks (or packet flows), and transmits the blocks to other edge caches or to clients. (4) Request processing service that provides navigation of the CDN to locate the content.

When a client requests or subscribes to digital media content, the request is propagated to the edge cache closest to the client. If the cache contains the requested content, the edge cache transmits the content to the client. If not, then the request is forwarded to the core data center, which sends the content to the client. If a content is cacheable, the edge cache fetches the content from the core data center (or alternatively, the core data center pushes cacheable content to the edge cache) using an appropriate caching strategy to serve future requests for the content.

5.2.2 Stream authentication in content distribution networks

Threats involved in distribution of content include masquerading attacks where an adversary poses as an edge cache or a core data center or a third-party provider that serves data for dynamically generated content to distribute malicious content (possibly infected with worms, viruses, etc.), compromising attacks where an adversary takes control of legitimate content providing hosts (edge cache/core data center/third-party provider) to inject malicious content and man-in-the-middle attacks where an adversary performs modification of content during transmission from core data center to the edge cache or from

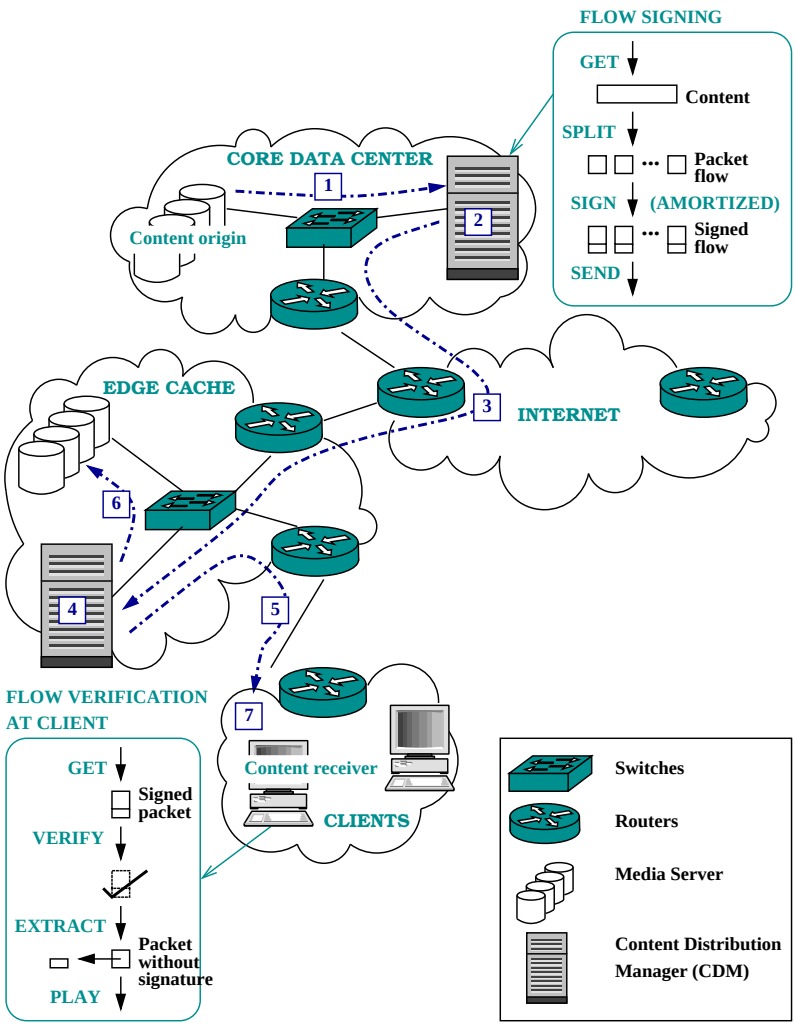

Figure 5.1: Architecture of a content distribution network.

the edge cache to the client or from the core data center to the client. Stream authentication can help prevent these attacks by providing the ability to sign and verify each block in the stream. All content originates at the core data center and the stream signing mechanism is implemented at the core CDM as part of its content processing service. We assume the existence of a public key infrastructure (PKI) responsible for generating certificates for the core CDM, and distributing the public key and certificate of the core CDM to all verifying entities. We do not delve into the details of roles and models of a PKI in this discussion. Referring to Figure 5.1, when a request arrives at the core CDM, the content processing service retrieves the content from the media server (*Step 1*). The core CDM then splits the content into a stream of blocks, signs each block (using a suitable signature amortization technique), places the authenticating information within the block (*Step 2*) and transmits the signed stream of blocks to the requesting entity (*Steps 3 and 5*). If the content is

not generated in real-time, the content processing service stores the signed stream at the media server to prevent redundant signing operations when subsequent requests arrive for the same content. The stream verification mechanism should be implemented by both the CDM at an edge cache and at the client machine. When a signed stream arrives at an edge cache, the content processing service at the CDM verifies individual blocks in the stream (*Step 6*), stores the signed content into the cache's media server (*Step 7*), and later, when a request arrives for the content sends the signed stream to the requesting client (*Step 8*). Verification of signed streams at edge caches ensures that packets failing verification are not forwarded to the requesting client thereby preventing unnecessary usage of bandwidth and processing time at the client machine. When a signed stream arrives at the client machine, the requesting application verifies each block in the stream and removes the authenticating information placed inside the block before it plays of the media content (*Step 4*).

To provide reasonable quality of service to clients of CDNs, a stream authentication mechanism must overcome the following challenges: (1) Delays in real-time content transmission can affect client applications such as stock traders that rely on timely delivery of live news feeds and stock quotes for critical decision making. As content is transmitted from the core data center to the client, delays are introduced by the operations of signing each block at the core data center and transmission delays from core data center to the client. (2) High per-block processing overhead can severely degrade content playback performance in devices like PDAs and cell phones with limited memory and processing power. Storage and computation requirements of verifying a signed block increases the block's processing overhead. (3) Loss of blocks can be high when distributing content using non-reliable transport protocols or through lossy channels like wireless transmission. (4) Available bandwidth in CDNs is limited. To provide good scalability and reduce transmission delays, content size should be minimized for a desired quality of service. Placing authenticating information in each block increases the content size and consequently bandwidth and time requirements to transmit the content. In the next section, we present a novel signature amortization technique using trapdoor hash functions that can be used to build efficient stream authentication mechanisms and overcome the aforementioned challenges.

5.3 Proposed signature amortization technique

In this section we present a novel trapdoor-hash based signature amortization technique that can be used for authenticating digital streams in CDN's, multicast systems and peer-to-peer networks.

5.3.1 Trapdoor hash-based signature amortization technique

The proposed signature amortization technique works by authenticating the initial block of a stream using a signature on trapdoor hash of the block's contents, and authenticating subsequent block of the stream by finding trapdoor collisions with the hash of the signed (initial) block. As long as the initial block containing the signature is reliably delivered (and verified), the verifier can authenticate any block in the stream by matching its trapdoor hash value with *any* previously computed trapdoor hash — all blocks in the stream hash to the same value, and thus, trapdoor hash of any arbitrary block in the stream can be used for comparison during block verification.

The stream $\mathcal{S}$ is denoted as a logically ordered sequence of blocks $\langle p_0, p_1, \dots \rangle$. The contents of each block p_i, is denoted by m_i. The sender possess a long-term (private, public) key pair (SK, PK) as well as a (trapdoor, hash) key pair (TK_0, HK_0). We assume existence of a public key infrastructure, where each entity is associated with a certificate signed by a certificate authority (CA) binding the entity's identity with its public key. The PKI is responsible for distribution and maintenance of public keys and certificates to all entities in the system.

The proposed signature amortization technique can be divided into two phases: Stream signing and stream verification. During the stream signing phase, the sender generates the authenticating information for each block in the stream as follows:

Initial block generation: The sender generates a signature σ using SK on the trapdoor hash $h_0 = TH_{HK_0}(m_0, r_0)$ of the contents m_0 of first block p_0 in the stream. The first block p_0 contains $\langle m_0, r_0, \sigma \rangle$.

Subsequent block generation: To sign subsequent blocks p_i $(i \geq 1)$ with content m_i, the signer generates a collision parameter r_i such that $TH_{HK_{i-1}}(m_{i-1}, r_{i-1}) = TH_{HK_i}(m_i, r_i)$ using trapdoor keys TK_i and TK_{i-1}. Note that the pairs (TK_{i-1}, HK_{i-1}) and (TK_i, HK_i), need not be different [cf. Section 3.2]. When ephemeral trapdoor and hash keys are used, each key pair is utilized for authenticating one block and then discarded. Block p_i contains $\langle m_i, r_i \rangle$ and optionally HK_i depending on whether an ephemeral trapdoor key is used for collision computation (i.e., $TK_i \neq TK_{i-1}$).

During the verification phase, the receiver checks the authenticity of each block in the stream as follows:

Initial block verification: When the receiver obtains the initial block p_0 of the stream, it extracts $\langle m_0, r_0, \sigma \rangle$ from the block, computes $h_0 = TH_{HK_0}(m_0, r_0)$ and verifies the signature σ on h_0 under PK.

Subsequent block verification: For each subsequent block, p_i $(i \geq 1)$ in the stream, the receiver parses the received block as $\langle m_i, r_i \rangle$ and computes the trapdoor hash of m_i as $h_i = TH_{HK_i}(m_i, r_i)$. It then checks whether h_i matches the trapdoor hash $h_j = TH_{HK_j}(m_j, r_j)$ of the contents m_j of an arbitrary block p_j that was received prior to p_i. If trapdoor hashes match, the verification is successful. Otherwise, the receiver sends an error message to the signer and aborts.

5.3.2 Features of the proposed signature amortization technique

We present a brief discussion on security and performance merits of the proposed trapdoor hash-based signature amortization technique for stream authentication.

Robustness Against Packet Loss: In the proposed technique, to compute the trapdoor hash value of the contents m_i of a block p_i $(i \geq 0)$, the verifier needs to know r_i which is contained within block p_i. Moreover, verification of each subsequent block p_i $(i \geq 1)$ depends on trapdoor hash value of *any* block p_j received prior to p_i. Thus, the proposed technique can tolerate arbitrary loss of blocks in the stream as long as the initial block containing the signature is reliably delivered to the receiver — any subsequent block can be verified by comparing its trapdoor hash value with the trapdoor hash value of the initial block. The proposed technique can use several techniques for reliable delivery of the signature block including re-transmissions, error-correcting codes, placement of signature in multiple blocks, or by transmitting the signature using a separate reliable channel [55, 82, 66].

Fast Signing and Verification Rates: The proposed technique requires the sender to maintain a single block in its buffer, which is used for computing trapdoor collisions for the current block. Also, since verification of each block is dependent on one other block, the verifier only needs to maintain content of a single block in its buffer. Thus, the proposed technique is highly suitable for authenticating real-time and delay sensitive streams, and resilient against denial of service (DoS) attacks that exploit buffer overflow. Computation costs at the sender side involve one signature generation for the initial block and one collision computation for subsequent blocks. At the receiver side, initial block authentication requires one signature verification, and authentication of subsequent blocks requires one trapdoor hash computation. Thus, computation costs at sender and receiver side remain constant per block.

Constant Per-Packet Communication Overhead: Communication overhead incurred by the initial block, p_0 includes the signature σ and the random string r_0 used in the computation of trapdoor hash of message m_0. Each subsequent block p_i contains a random string (collision result) r_i (and an optional ephemeral hash key HK_i depending on the choice of the trapdoor hashing scheme) used in the computation of trapdoor hash of message m_i. Thus, each block incurs constant communication overhead and there is no redundant placement of the same authenticating material in multiple blocks.

Prevention of Packet Modification: Prevention of content modification using the proposed signature amortization technique relies on the forgery resistance of the signature scheme (used during initial packet authentication) and trapdoor hashing scheme possessing the following properties: (1) Collision forgery resistance; (2) Key exposure resistance; (3) Semantic Security. The goal of the adversary is to modify contents of an existing block or insert a new block in the stream so that the malicious block p'_i $(i = 0, 1, \dots)$ in the stream of blocks $\langle p_0, p_1, \dots \rangle$ passes the verification procedure at the receiver side. We assume the channel is insecure and the adversary is able to capture and store contents of all blocks that comprise the stream. If the trapdoor hash function is collision resistant, creating a malicious initial block p'_0 requires the adversary to forge a signature on the contents of the malicious block p'_0. Thus, given a collision resistant trapdoor hash function and a forgery resistant signature scheme, forging an initial block would be computationally infeasible. Forging a subsequent block p'_i $(i \geq 1)$ in the stream requires the adversary to forge a collision between trapdoor hash of some previous block in the stream and the contents of the malicious block. Given a collision resistant trapdoor hash function, collision forgery is computationally infeasible without knowledge of the trapdoor key. Moreover, by using key exposure-free trapdoor hash functions we ensure that given a pair of messages whose trapdoor hash values result in a collision, the adversary is not able to extract the trapdoor key of the sender.

5.4 A discrete log-based signature amortization scheme, DL-SA

We now present a discrete-log based instantiation of the proposed technique for signature amortization using trapdoor hash functions. The proposed scheme uses the trapdoor hashing scheme DL-MTH described in Chapter 3.

During initialization, given the security parameter λ as input, all entities choose and agree upon common system public parameters $\mathsf{params} = \langle p, q, \alpha, H, G \rangle$, where p and q are 1024-bit and 160-bit primes, respectively, $q \mid p - 1$, α is an element of order q in $\mathbb{Z}_p^*$ and

$H, G : \{0,1\}^* \mapsto \mathbb{Z}_q^*$ are cryptographic hash functions. The sender's long-term (private, public) key pair is given by $(SK, PK) = (x, X)$, where $x \in_R \mathbb{Z}_q^*$ and $X = \alpha^x \in \mathbb{Z}_p^*$. The sender's long-term (trapdoor, hash) key pair is given by $(TK_0, HK_0) = (y_0, Y_0)$, where $y_0 \in_R \mathbb{Z}_q^*$ and $Y_0 = \alpha^{y_0} \in \mathbb{Z}_p^*$.

The stream $\mathcal{S}$ is partitioned into segments $s_0, s_1, \ldots$ with each segment s_i containing multiple blocks $p_{i,0}, p_{i,1}, \ldots$. The number of blocks in each segment need not be the same to accommodate for time-varying generation of streams [82]. However, for simplicity we assume that each segment contains n blocks

We now consider the authentication of the first segment s_0 in the stream containing the blocks $p_0, p_1, \ldots, p_{n-1}$ (for ease of notation, we drop the subscript indicating the segment number). During the stream signing phase, the sender generates the authenticating information for each block in the first segment of the stream as follows:

Initial block generation: The sender generates a signature σ using SK on the trapdoor hash of the contents m_0 of first block p_0 in the stream as follows:

1. Choose an ephemeral private key $k_0 \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral public key $r_0 = \alpha^{k_0} \in \mathbb{Z}_p^*$.
2. Compute the trapdoor hash of m_0 as $TH_{Y_0}(m_0, r_0) = \alpha^{h_{m_0} + y_0 r_0} \mod q$, where $h_{m_0} = H(m_0 || Y_0)$, using its long term hash key Y_0.
3. Solve for t in the equation: $t \equiv k_0 + xG(TH_{Y_0}(m_0, r_0) || m_0 || r_0) \mod q$ (this is the well-known `DL-Schnorr` [72] signature scheme).
4. Signature on m_0 is given by $\sigma = \langle t, r_0 \rangle$.

The signer appends σ to the contents of the first block p_0 to generate the signed block $p_0 = \langle m_0, \sigma \rangle$.

Subsequent block generation: To sign subsequent blocks p_i $(i \geq 1)$ with content m_i, the signer computes the following:

1. Choose an ephemeral trapdoor key $y_i \in_R \mathbb{Z}_q^*$ and compute the corresponding ephemeral hash key $Y_i = \alpha^{y_i} \in \mathbb{Z}_p^*$. Store the pair (y_i, Y_i).
2. Compute $h_{m_i} = H(m_i || Y_i)$ and using trapdoor key y_0 and ephemeral key y_i, solve for r_i in the equation: $r_i = y_i^{-1}(h_{m_0} - h_{m_i} + y_0 r_0) \mod q$.

The signer appends r_i and Y_i to the contents of the subsequent block p_i to generate the signed subsequent block $p_i = \langle m_i, r_i, Y_i \rangle$.

During the verification phase, the receiver checks the authenticity of the each block in the first segment of the stream as follows:

Initial block verification: When the receiver obtains the initial block p_0 of the stream, it extracts $\langle m_0, \sigma \rangle$ from the block and verifies $\sigma = \langle t, r_0 \rangle$ as follows:

1. Look up the long-term hash key Y_0 and public key X of sender from a publicly available directory and store (X, Y_0). Compute $h_{m_0} = H(m_0||Y_0)$.
2. Compute $TH_{Y_0}(m_0, r_0) = \alpha^{h_{m_0}} Y_0^{r_0} \mod q$ and $h = G(TH_{Y_0}(m_0, r_0)||m_0||r_0)$.
3. Compute $r' = \alpha^t X^{-h} \mod q$. If $r' \neq r_0 \mod q$, output *Invalid* and abort.
4. Otherwise, output *Valid* and store $TH_{Y_0}(m_0, r_0)$ in cache.

Subsequent block verification: Depending on whether a subsequent block needs to be individually verified or batch verified along with previously cached blocks, verification proceeds in one of the two following ways:

Individual block verification: When we desire individual verification of a subsequent block, p_i $(i \geq 1)$, the receiver parses the received block p_i as $\langle m_i, r_i, Y_i \rangle$, stores Y_i, and verifies p_i as follows:

1. Retrieve $TH_{Y_0}(m_0, r_0)$ from cache and compute $h_{m_i} = H(m_i||Y_i)$.
2. Compute $TH_{Y_i}(m_i, r_i) = \alpha^{h_{m_i}} Y_i^{r_i} \mod q$.
3. Check whether $TH_{Y_0}(m_0, r_0) = TH_{Y_i}(m_i, r_i)$. If check fails, output *Invalid*.
4. Otherwise, output *Valid*.

Batch block verification: When we desire batch verification of multiple cached blocks, $p_a, p_{a+1}, \ldots, p_{a+l}$ $(l \geq 2)$, the receiver parses each the received block p_i as $\langle m_i, r_i, Y_i \rangle$, stores each Y_i, and batch verifies all blocks in its cache as follows:

1. Compute $h_{m_i} = H(m_i||Y_i)$ for $i = a, a+1, \ldots, a+l$ and retrieve $TH_{Y_0}(m_0, r_0)$ from cache.
2. Compute $h = \sum_{i=0}^{l} h_{m_{a+i}} \mod q$
3. Check whether $\alpha^h \prod_{i=0}^{l} Y_{a+i}^{r_{a+i}} = \prod_{i=0}^{l} TH_{Y_0}(m_0, r_0) \mod q$. If check fails, output *Invalid*.
4. Otherwise, output *Valid*.

The procedure is repeated for authenticating blocks within subsequent segments s_i $(i \geq 1)$ of the stream with the following differences. The signer generates the signature, $\sigma_{i,0} =$

$\langle t_{i,0}, r_{i,0} \rangle$ on the trapdoor hash of $m_{i,0}$ (contents of the first block $p_{i,0}$ of the segment s_i, $i \geq 1$) following the initial block generation procedure of DL-SA. To authenticate a subsequent block $p_{i,j}$ ($j \geq 1$), the signer skips step 1 of the subsequent block generation procedure and computes the collision parameter $r_{i,j}$ of the message $m_{i,j}$ (contained in $p_{i,j}$) as $r_{i,j} = y_i^{-1}(h_{m_{i,0}} - h_{m_{i,j}} + y_0 r_{i,0}) \mod q$ using the stored trapdoor key y_i. The signer appends $r_{i,j}$ to the contents of the subsequent block $p_{i,j}$ to generate the signed subsequent block $p_{i,j} = \langle m_{i,j}, r_{i,j} \rangle$ (that does not contain a hash key). The verifier verifies the subsequent block using the stored hash key Y_i. This repeated use of the (trapdoor, hash) key pairs $(y_0, Y_0), (y_1, Y_1), \ldots, (y_n, Y_n)$ to authenticate the blocks in subsequent segments of the stream saves computation at the sender and receiver, in addition to drastically reducing the per-block communication overhead.

During batch verification of multiple packets $p_a, p_{a+1}, \ldots, p_{a+l}$, the cached blocks need not be contiguous or sequential (the indices do not indicate the order in which packets were sent, but instead, only serve to indicate the number of blocks in the receiver's cache). Batch verification only requires that the receiver is able to cache blocks regardless of their logical sequence at the sender and can tolerate arbitrary loss of intermediate blocks and non-sequential arrival of blocks.

5.5 Analysis of the DL-SA signature amortization scheme

We now present a thorough analysis of the proposed signature amortization scheme DL-SA including its correctness, security and performance evaluation.

5.5.1 Correctness

The proposed discrete log-based signature amortization scheme, DL-SA is correct if the initial block p_0 and all subsequent blocks p_i in the stream pass the verification procedure at the receiver, provided: (1) All entities honestly choose and agree upon the system public parameters $\mathsf{params} = \langle p, q, \alpha, H, G \rangle$; (2) The sender honestly executes the key generation algorithm to generate its (private, public) key pair (SK, PK) and (trapdoor, hash) key pair (TK, HK); (3) The sender honestly executes the initial block signing procedure to generate σ on trapdoor hash of m_0; (4) The sender honestly computes the trapdoor collision parameter r_i, and hash value h_{m_i} to authenticate each subsequent block p_i in the stream.

Proposition 5.5.1. *The proposed DL-based signature amortization technique,* DL-SA *is correct.*

Proof: We prove the correctness of the proposed DL-based signature amortization technique, by showing that the initial packet p_0 and subsequent packets p_i ($i \geq 1$) pass the verification procedure at the receiver provided the aforementioned conditions hold true. During verification of the initial signature $\sigma = \langle t, r_0 \rangle$ on m_0 we observe the following:

$$\alpha^t X^{-h} = \alpha^{(k_0 + xh - hx)} = \alpha^{k_0} = r_0 \tag{5.1}$$

where $h = G(TH_{Y_0}(m_0, r_0)||m_0||r_0)$. Thus, σ passes the verification procedure under public key X.

When a subsequent packet p_i containing m_i, r_i and Y_i is individually verified we observe the following:

$$\begin{aligned}
TH_{Y_i}(m_i, r_i) &= \alpha^{h_{m_i}} Y_i^{r_i} \\
&= \alpha^{h_{m_i} + y_i r_i} \\
&= \alpha^{h_{m_i} + y_i y_i^{-1}(h_{m_0} - h_{m_i} + y_0 r_0)} \quad (\text{Since } r_i = y_i^{-1}(h_{m_0} - h_{m_i} + y_0 r_0) \mod q) \\
&= \alpha^{h_{m_i} + h_{m_0} - h_{m_i} + y_0 r_0} \\
&= \alpha^{h_{m_0} + y_0 r_0} \\
&= TH_{Y_0}(m_0, r_0) \mod q
\end{aligned}$$

Thus, p_i passes the verification procedure at the receiver. What remains to be shown is that cached packets $p_a, \ldots, p_{a+l}$ ($l \geq 2$) pass the batch verification procedure at the receiver. We know that for each $i \geq 1$, $TH_{Y_i}(m_i, r_i) = TH_{Y_0}(m_0, r_0)$. During verification of packets $p_a, \ldots, p_{a+l}$ we observe the following:

$$\begin{aligned}
\alpha^h \prod_{i=0}^{l} Y_{a+i}^{r_{a+i}} &= \alpha^{\sum_{i=0}^{l} h_{m_{a+i}}} \prod_{i=0}^{l} Y_{a+i}^{r_{a+i}} \\
&= (\alpha^{h_{m_a}} Y_a^{r_a})(\alpha^{h_{m_{a+1}}} Y_{a+1}^{r_{a+1}}) \ldots (\alpha^{h_{m_{a+l}}} Y_{a+l}^{r_{a+l}}) \\
&= \prod_{i=0}^{l} \alpha^{h_{m_0}} Y_0^{r_0} \\
&= \prod_{i=0}^{l} TH_{Y_0}(m_0, r_0) \mod q
\end{aligned}$$

Thus, packets $p_a, \ldots, p_{a+l}$ pass the batch verification procedure at the receiver. □

5.5.2 Security

Security of the proposed DL-based signature amortization scheme depends on the following two factors: (1) forgery resistance of the signature scheme used for signing the initial

block in the stream and (2) collision forgery, ephemeral collision forgery and key exposure resistance of the trapdoor hashing scheme used for signing each subsequent block in the stream. Security of the Schnorr signature scheme against forgery is based on the well-known Theorem 2.2.1. In Chapter 3 we show that the DL-MTH trapdoor hashing scheme is collision forgery resistant, ephemeral collision forgery resistant and key exposure resistance. We now describe an adversarial model that we use to demonstrate that the proposed DL-SA scheme is secure against adaptive chosen message attack under the discrete log assumption.

Adversarial Model: Consider a game $G_{\mathcal{A}}^{DLSA}(\lambda)$, where λ is a security parameter. The game is played by a PPT adversary $\mathcal{A}$ with an honest entity e. The game $G_{\mathcal{A}}^{DLSA}(\lambda)$ is initialized as follows. On security parameter λ as input, all entities choose and agree upon the system public parameters params. Entity e generates its (private, public) key pair (x, X) and its (trapdoor, hash) key pair (y_0, Y_0), and registers (X, Y_0) in a publicly available directory. $\mathcal{A}$ is provided with the target public key, X and hash key, Y_0 of the honest entity e. The adversary is also given access to a block signing oracle $\mathcal{O}_S$ that generates signatures on data blocks that are valid under X and Y_0. The adversary proceeds adaptively and request the oracle $\mathcal{O}_S$ for signed data blocks. When $\mathcal{A}$ queries $\mathcal{O}_S$ with input m, $\mathcal{O}_S$ returns $\langle m, \sigma \rangle$, where $\sigma = \langle t, r \rangle$ is a Schnorr signature on $h = G(TH_{Y_0}(m, r)||m||r)$ that is valid under public key X following the initial block verification procedure of DL-SA. When $\mathcal{A}$ queries $\mathcal{O}_S$ with input $\langle p_0, p_1, \ldots, p_n, m_{n+1} \rangle$, where $p_0 = \langle m_0, t_0, r_0 \rangle$ and $p_i = \langle m_i, r_i, Y_i \rangle$ $(i = 1, \ldots, n)$, $\mathcal{O}_S$ verifies each block following the verification procedures described in Section 5.4, and if all blocks are valid, returns $p_{n+1} = \langle m_{n+1}, r_{n+1}, Y_{n+1} \rangle$ such that $TH_{Y_0}(m_0, r_0) = TH_{Y_{n+1}}(m_{n+1}, r_{n+1})$.

The goal of the adversary is to output one or more of the following forgeries:

Type I: $\mathcal{A}$ outputs a forged initial block p_0^F containing $\langle m_0^F, \sigma^F \rangle$ such that m_0^F was never submitted as a query to $\mathcal{O}_S$ and $\sigma^F = t_0^F, r_0^F$ passes DL-SA's initial block verification procedure under public key X.

Type II: $\mathcal{A}$ outputs a forged sequence of blocks $\langle p_0^F, p_1^F, \ldots, p_n^F \rangle$ such that the following hold: (1) m_n^F was never submitted as part of a query to $\mathcal{O}_S$; (2) $\langle p_0^F, p_1^F, \ldots, p_{n-2}^F, m_{n-1}^F \rangle$ was submitted as a query to $\mathcal{O}_S$ and $p_{n-1}^F = \langle m_{n-1}^F, r_{n-1}^F, Y_{n-1}^F \rangle$ was received as a response, and (3) p_0^F passes DL-SA's initial block verification procedure under public key X, and $\langle p_1^F, \ldots, p_n^F \rangle$ pass DL-SA's subsequent block verification procedure under hash key Y_0.

The advantage of the adversary, $Adv_{\mathcal{A}}^{DLSA}(\lambda)$, is the probability that $\mathcal{A}$ successfully outputs a Type I or a Type II forgery after proceeding adaptively with oracles $\mathcal{O}_H$, $\mathcal{O}_G$ and $\mathcal{O}_S$. We say that the proposed DL-based signature amortization technique is secure against adaptive chosen message attack if $Adv_{\mathcal{A}}^{DLSA}(\lambda)$ is negligible for all PPT algorithms $\mathcal{A}$ of polynomial time complexity in the security parameter λ.

Based on Theorems 2.2.1, 3.4.1, and 3.4.2, we now prove that the signature amortization scheme, `DL-SA` is secure against an adaptive chosen message attack.

Theorem 5.5.2. *The proposed signature amortization scheme,* `DL-SA` *is secure against adaptive chosen message attack under the discrete log assumption.*

Proof: Given $\mathcal{A}$, we can construct a PPT algorithm $\mathcal{B}$ that breaks the discrete log problem. Algorithm $\mathcal{B}$ acts as a simulator that runs $\mathcal{A}$, answering $\mathcal{A}$'s oracle queries, to convert a successful attack by $\mathcal{A}$ on the proposed signature amortization scheme `DL-SA` into an attack on the discrete log of a target public key $PK = X$. $\mathcal{B}$ simulates the hash functions H and G as random oracles $\mathcal{O}_H$ and $\mathcal{O}_G$, respectively, and a stream block signature oracle O_S that answers queries of $\mathcal{A}$.

$\mathcal{B}$ chooses $a \in_R \mathbb{Z}_q^*$, computes $Y_0 = X^a \mod p$ and assigns X and Y_0 as the public key and hash key, respectively, of an arbitrary entity e. $\mathcal{B}$ initializes an empty list M_S to store the queries of $\mathcal{A}$ to oracle $\mathcal{O}_S$ and the corresponding (simulated) responses of $\mathcal{B}$. When $\mathcal{A}$ queries $\mathcal{O}_H$ with a message m, $\mathcal{B}$ returns h if $H(m)$ is defined (that is, $\exists h$ such that $h = H(m)$). Otherwise, $\mathcal{B}$ chooses $h \in_R \mathbb{Z}_q^*$, sets $H(m) = h$, stores h as the hash entry for $H(m)$ and returns h to $\mathcal{A}$. Similarly, $\mathcal{B}$ simulates responses to $\mathcal{A}$'s queries to hashing oracle O_G (modelling the hash function G). When $\mathcal{A}$ queries $\mathcal{O}_S$ with input m, $\mathcal{B}$ does the following:

1. Checks whether $\langle m, \cdot \rangle \in M_S$. If so, retrieves that entry, parses it as $\langle m, \sigma \rangle$ and returns σ to $\mathcal{A}$.

2. Otherwise, chooses $g, t \in_R \mathbb{Z}_q^*$ and computes $r = \alpha^t X^{-g} \mod q$.

3. Chooses $g \in_R \mathbb{Z}_q^*$, sets $H(m||r) = h$ and stores h as the hash entry for $H(m||r)$.

4. Computes $TH_{Y_0}(m, r) = \alpha^{H(m||Y_0)} Y_0^r \mod q$, sets $G(TH_{Y_0}(m, r)||m||r) = g$ and stores g as the hash entry for $G(TH_{Y_0}(m, r)||m||r)$.

5. Stores $\langle m, t, r \rangle$ at the next available location in M_S.

6. Returns $\langle t, r \rangle$ to $\mathcal{A}$.

When $\mathcal{A}$ queries $\mathcal{O}_S$ with input $\langle p_0, p_1, \ldots, p_n, m_{n+1}\rangle$, $\mathcal{B}$ does the following:

1. Checks whether $\langle p_0, \cdot, \cdots\rangle \in M_S$ (where, $p_0 = \langle m_0, t_0, r_0\rangle$). If not $\mathcal{B}$ returns $\perp$ and aborts. $\mathcal{B}$ retrieves that entry and parses it as $\langle p_0, p'_1, \ldots, p'_i\rangle$. If $i < n$ the query is invalid, and $\mathcal{B}$ returns $\perp$ and aborts. Otherwise, rearrange $\langle p_1, \ldots, p_n\rangle$ to get $\langle \bar{p}_1, \ldots, \bar{p}_n\rangle$ such that $\forall s < t$, $\bar{r}_s \leq \bar{r}_t$ (i.e., the p_i's are arranged in increasing order of their r component). Similarly, rearrange $\langle p'_1, \ldots, p'_n\rangle$ to get $\langle \bar{p}'_1, \ldots, \bar{p}'_n\rangle$. For $j = 1, \ldots n$, check whether $\bar{p}'_j = \bar{p}_j$. If not, query is invalid, and $\mathcal{B}$ returns $\perp$ and aborts.

2. If $i > n$, returns p'_{n+1}. Otherwise, $\mathcal{B}$ chooses $h_{n+1}, r_{n+1} \in_R \mathbb{Z}_q^*$, retrieves $h_0 = H(m_0||Y_0)$ and if no such h_0 exists aborts. $\mathcal{B}$ computes $Y_{n+1} = \alpha^{(h_0 - h_{n+1})r_{n+1}^{-1}} Y_0^{r_0 r_{n+1}^{-1}} \mod p$. Set $H(m_{n+1}||Y_{n+1}) = h_{n+1}$ and stores h_{n+1} as the hash entry for $H(m_{n+1}||Y_{n+1})$.

3. Concatenates $p_n = \langle m_{n+1}, r_{n+1}, Y_{n+1}\rangle$ to the entry in M_S corresponding to $\langle p_0, \cdot, \cdots\rangle$.

4. $\mathcal{B}$ returns $p_n = \langle m_{n+1}, r_{n+1}, Y_{n+1}\rangle$ to $\mathcal{A}$.

Eventually $\mathcal{A}$ outputs a Type I or Type II forgery. When $\mathcal{A}$ outputs a Type I forgery of the form p_0^F containing $\langle m_0^F, \sigma^F\rangle$, this implies that $\langle t_0^F, r_0^F\rangle$ is a forged Schnorr signature on m_0^F. Based on Theorem 2.2.1, $\mathcal{B}$ outputs the discrete log x of the target public key X. When $\mathcal{A}$ outputs a Type II forgery of the form $\langle p_0^F, p_1^F, \ldots, p_n^F\rangle$, this implies that one of the following must hold true: (1) Assume that $\mathcal{A}$ submitted m as a query to O_S to receive $\langle t, r\rangle$ as a response. Also, assume that $\mathcal{A}$ submitted $\langle p, m'\rangle$, where $p = \langle m, t, r\rangle$, as a query to O_S to receive $\langle m', r', Y'\rangle$ as a response. $\mathcal{A}$ uses $\langle m, m', r, r', Y_0, Y'\rangle$ (where, $TH_{Y_0}(m, r) = TH_{Y'}(m', r')$) to compute the discrete log y_0 of Y_0 (performs a key exposure) and using y_0 generates the forged block p_n^F. (2) $\mathcal{A}$ generates the pair $\langle r_n^F, Y_n^F\rangle$ such that $TH_{Y_0}(m_0^F, r_0^F) = TH_{Y_n^F}(m_n^F, r_n^F)$ (i.e., performs an ephemeral collision forgery). In either case, $\mathcal{B}$ computes the discrete log x of the target public key X as $x = y_0 - a \mod q$. □

5.5.3 Performance

In this section we present an experiment-based performance evaluation of the proposed scheme, `DL-SA`. First, we provide a brief description of the system setup used for our experiments that simulates a wider-scale approach for real-world content authentication. Next, we describe our approaches for improving the efficiency of the proposed scheme in our simulations. Finally, we compare the performance of the proposed scheme, `DL-SA` with the

WL [82] scheme and the CHFS [71] schemes based on results obtained through our experiments. We choose the WL and CHFS schemes for comparison as they are the only known signature amortization techniques in the literature that exhibit features that are comparable to the proposed scheme, which include, the ability to independently verify each block in the stream (and thus, are robust against arbitrary loss of intermediate blocks) and the capability to authenticate real-time streams. Shortcomings of other schemes in providing the independent verifiability and real-time authentication support are elaborated in Section 2.2.4.

System Set-up: Our performance comparison involves experiments to test the performance of each scheme which include the average per-block signing and verification timings, the average number of additional bits that needs to be appended to a block to provide authentication, and the size of the public key (representing the storage overhead) that is required for verifying the authenticity of each block in the stream. We conducted our experiments on a single machine with Intel(R) Core(TM) i7 processor with 6 GiB RAM running 64-bit Windows(R) 7. For each scheme, we implemented a stream signing mechanism and a stream verification mechanism using the Java.Security package in Java Standard Edition 6 using the Eclipse integrated development environment. The signing and verification mechanisms are each modeled as a class with methods (or subroutines) for the necessary cryptographic operations.

On a wider-scale test-bed, the signing and verification mechanisms will be implemented as shared libraries or objects (possibly remote) at the server-side and client-side as part of the operating system to allow loading the subroutines of a library into multiple applications at runtime. The approach of using shared libraries for content authentication is similar to Microsoft's Authenticode(TM) technology for the Internet Explorer that is used for signing and verifying executable software distributed over the Internet. The flexibility provided by dynamic linking of shared libraries will permit multiple flavors of server-side CDM software and client-side stream playback software to utilize the authentication mechanism. Another alternative is to implement the stream authentication procedure as an extension to the transport layer protocols that are used to packetize the stream before transmission. However, this approach only permits signing a stream at the time of transmission and does not allow offline signing (for non-real time generated streams) and storage of a signed stream for future replay (or for caching). An application layer approach (using shared libraries) to signing and verifying content provides more flexibility compared to a transport layer mechanism.

In our experiments, we represent the content as a large file (between 50 to 100 MiB

in size) that is given to an instance of stream signing class as input during runtime. A method in the signing class splits the file into blocks of equal length, groups the blocks into equal sized segments and stores the blocks in a two-dimensional array, where each column of the array represents a segment containing multiple blocks. This array of blocks is given as input to the block signing method that outputs the signed blocks which are, in turn, stored in a two dimensional output array. To simulate transmission losses, random entries in the array of signed blocks are replaced with a null value. Next, the array of signed blocks (representing the signed stream) is given as input to an instance of the stream verification class that verifies each block in the array. Our system setup of using Java classes to represent the stream signing and verification mechanisms is a sufficiently accurate simulation of a real-world deployment scenario and is adequate for the purposes of performance analysis.

Speed-up of `DL-SA`: In the `DL-SA` scheme, exponentiation is the most expensive operation required to sign and verify each block. The exponentiation operations can be sped up using the Lim and Lee (LL) technique [51] and the simultaneous multi-exponentiation [60] at the expense of requiring the sender and receiver to store pre-computed values. For an exponent e of size 160-bits and a base α of size 1024-bits, the standard square-and-multiply technique requires 240 multiplications on average. In contrast, the LL method requires only 24 multiplications to compute α^e with storage of 1020 1024-bit values (which equates to 127.5 KiB). For fixed bases $\alpha_1, \alpha_2, \ldots, \alpha_k$, the simultaneous multi-exponentiation technique can be used to efficiently compute $\alpha_1^{e_1}\alpha_2^{e_2}\cdots\alpha_k^{e_k}$ using 299 multiplications with storage of 2^k values (the pre-computation requires $(2^k - 2)$ multiplications, but only needs to be performed once).

We employ the LL and simultaneous multi-exponentiation techniques to optimize the cost of computing exponentiations and thus, improve the efficiency of the proposed scheme. Depending on how the optimization techniques are used, we obtain two variants of `DL-SA`. The first variant (called the `DL-SA.1`) uses the LL technique for all exponentiation operations at the sender and receiver. The second variant (called the `DL-SA.2`) uses LL technique to compute exponentiations at the sender, the LL technique to compute exponentiations during individual block verification and simultaneous multi-exponentiation during batch verification of blocks.

Performance Comparison: Table 5.1 shows the performance comparison of both variants of the proposed scheme against the `WL` and `CHFS` schemes. Our test-bed is used to simulate each scheme. In Table 5.1, the term x denotes the cost of modular exponentiation with 160-bit exponent in the DL-based cryptosystem, e denotes the cost of RSA encryption

Table 5.1: Performance comparison of proposed signature amortization scheme, DL-SA with existing schemes.

Scheme	Sig. Size (Bits)	Signing Cost		Verification Cost				PK size (Bits)
				Individual		Batch		
		Opns	Timing (ms)	Opns	Timing (ms)	Opns	Timing (ms)	
WL.1 [82]	704	$0.06d$ + $1.94h$	0.39	$0.06e$ + $0.88h$	0.01	N/A	N/A	1024
WL.2 [82]	1152	$0.06d$ + $1.94h$	0.02	$0.06e$ + $0.88h$	0.01	N/A	N/A	2048
CHFS [71]	2320	$0.03d$ + $349.22h$	1.74	$184h$	0.83	N/A	N/A	720
DL-SA.1	160	$0.03x$ + $1h$	0.03	$2.06x$ + $1h$	0.21	$0.06x$ + $\frac{l+1}{l}x+h$	0.16 to 0.11	1024
DL-SA.2	160	$0.03x$ + $1h$	0.03	$2.06x$ + $1h$	0.21	$0.06x$ + $\frac{2.49}{l}x+h$	0.11 to 0.09	1024

(i.e., modular exponentiation with 16-bit exponent), d denotes the cost of RSA decryption (i.e., modular exponentiation with 1024-bit exponent), h is the cost of cryptographic hash computation and l is the number of blocks that are batch verified in the proposed DL-SA scheme ($3 \geq l \geq 32$ for DL-SA.1 and $20 \geq l \geq 32$ for DL-SA.2). All computation, storage and communication costs are based on a security benchmark of 1024-bits.

For the WL scheme, we divide the stream into blocks of 1024-bytes in size, group 16 blocks into a segment, and amortize a signature over each segment as suggested by Wong et al. [82]. The first variant WL.1 uses the RSA signature scheme and the second variant WL.2 uses the eFFS signature scheme [82]. For the CHFS we divide the stream into blocks of 240-bytes in size, group 32 blocks into a segment, and use 36-time key pairs to amortize a RSA signature over each segment as suggested by Rohatgi et al. [71]. The signature size represents the size of the authenticating information contained in each block. Signing and verification costs represents the average cost for signing and verifying each block. The columns showing the number of operations required to sign and verify each block contain only the most significant (expensive) operations. For the DL-SA scheme, we assume that each segment contains 32 blocks, each of size 1024-bytes. The signature size corresponds to subsequent segments. All public keys necessary to verify the entire stream are transmitted in the first segment using an additional 1024-bits per block and not transmitted for subsequent segments.

The performance analysis shows that the DL-SA scheme incurs the smallest per-block communication overhead and provides the fastest block signing rates compared to the WL

and CHFS schemes. Although both variants of the WL scheme provides faster verification compared to the proposed scheme, they suffer from several problems which include variable communication overhead per-block, the inability to support authentication of multiple simultaneous streams, and introducing sender-side delay to maintain reasonable costs (see [71] for details). The DL-SA scheme does not suffer from any of the drawbacks of the WL scheme. The most efficient variant of the WL scheme, WL.2 uses the largest secret key (131072-bits) and the largest public key compared to the other schemes. The CHFS scheme also overcomes all drawbacks of the WL scheme but incurs more overhead compared to DL-SA in terms of signing cost, verification cost and per-block signature size. The DL-SA scheme provides the best balance between features and performance compared to both WL and CHFS schemes.

5.6 Summary

Mechanisms of stream authentication in content distribution networks help prevent masquerading attacks and malicious modification of content during transmission. However, efficient authentication of live, on-demand content is a challenging task and requires fast signing and verification, tolerance against transmission loss and small per-block communication overhead. We presented a novel trapdoor hash-based signature amortization technique that meets these challenges to provide efficient authentication of delay-sensitive and real-time streams in content distribution, multicast and peer-to-peer networks. Our signature amortization technique works by authenticating the initial block in a stream using a signature on the trapdoor hash of the block contents and authenticating subsequent blocks of the stream by finding trapdoor collisions with the hash of the signed initial block. We demonstrated the efficacy of the proposed technique to build practical instances by presenting a discrete log-based instantiation of the proposed technique called DL-SA. The DL-SA scheme was designed to allow both individual verification of a single block in the stream or batch-verification of multiple blocks to reduce the average verification cost per block. We provided a detailed analysis of the proposed scheme including its correctness, security and performance. We demonstrated that the DL-SA scheme incurs the smallest per-block communication overhead and provides the fastest block signing rates compared to existing stream authentication schemes of Wong et al. [82] and Rohatgi et al. [71] that also provide the ability to independently verify each block in the stream.

Chapter 6

Conclusion and future work

6.1 Thesis contributions

In this dissertation, we presented authentication schemes that can be used to secure distributed applications against various threats, including malicious manipulation of data, and adversaries masquerading as legitimate entities. To this end, we developed techniques to construct proxy signatures and signature amortization schemes using trapdoor hash functions. The proposed proxy signature scheme is designed to authenticate agents acting on behalf of users in agent-based computing systems. The proposed signature amortization scheme is designed to authenticate each block comprising a stream in content distribution systems. Our performance analysis shows that for a desired level of security, the proposed schemes outperform previous schemes, both in computation cost, and communication overhead. To summarize, we make the following contributions:

- The goal of this dissertation is to present new techniques to build various authentication schemes that use the collision finding property of trapdoor hash functions to authenticate messages. Our motivation for using trapdoor collisions for message authentication stems from the following observation: When a trapdoor hash function is used within a hash-then-sign signature scheme, it permits the party with knowledge of the trapdoor to *re-use* the signature value to authenticate other messages of choice by finding collisions between the hash of the original signed-message and the new message that needs to be signed. To achieve our goal, we developed a key-exposure and collision-resistant discrete log-based trapdoor hash function that, unlike existing trapdoor hashing schemes [3, 4, 26, 44, 74], is suitable for use as a building block to construct authentication schemes that use the process of computing hash collisions using a trapdoor key, rather than generating conventional signatures using a secret key, to authenticate messages. The proposed scheme is designed to ensure that it is computationally infeasible for a third party to compute additional hash collisions with the knowledge of a message pair whose trapdoor hash values are equal. The proposed scheme is resistant to collision forgery and key exposure under the discrete log assumption. The proposed scheme is more efficient compared to existing key-

exposure-free trapdoor hashing schemes in terms of computation overhead during trapdoor hash and collision computation.

- Agent-based computing is a popular approach for designing large-scale and complex distributed systems [40]. Agents are software instances that can be delegated by a user (individual or a company) to carry out operations on its behalf. To secure agent-based computing systems against malicious attacks, users have to establish trust relationships with agents which includes their security, reliability, availability, and business continuity guarantees. Moreover, sensitive information stored and processed by agents need to be protected from exposure, alteration and corruption. We study the problem of providing efficient authentication of agents in agent-based computing environments. To this end, we proposed a simple and elegant technique to construct provably secure proxy signature schemes using trapdoor hash functions that can be used to authenticate and authorize agents acting on behalf of users in agent-based computing systems. Unlike most common proxy signature schemes [13, 47, 43], the proxy signing key pair is not derived from the delegator's signature during the proxy delegation phase. Instead of generating a signature on the message in the traditional sense, the proxy uses its trapdoor key (known exclusively to itself) to find a trapdoor collision between the trapdoor hash of the warrant and the given message. We demonstrated the effectiveness of our approach for creating practical instances by constructing a discrete log-based instantiation of the proposed generic technique, called the `DL-TPS` scheme. The `DL-TPS` yields superior performance in terms of proxy signature verification among the schemes in [13, 39, 43, 47, 57, 69, 83] and produces the smallest proxy signatures compared to schemes in [13, 43, 47, 48, 57, 69]. The proposed scheme is resistant to forgery attacks under the discrete log assumption.

- In networks designed to distribute content, mechanisms for stream authentication can help prevent malicious modification of content during transmission and attacks where adversaries masquerade as legitimate content providers to distribute malicious content. Efficient authentication of live and on-demand content requires fast signing and verification, tolerance against transmission loss and small per-block communication overhead. We developed a novel trapdoor hash-based signature amortization technique that meets these challenges to provide efficient authentication of delay-sensitive and real-time streams in content distribution, multicast and peer-to-peer networks. The basic idea behind the proposed signature amortization technique

is to authenticate the initial block of a stream using a signature on the trapdoor hash of the block contents and authenticate subsequent blocks of the stream by finding trapdoor collisions with the hash of the signed initial block. We also constructed a discrete log-based instantiation of the proposed technique, called `DL-SA`. The `DL-SA` scheme allows both individual verification of a single block in the stream or batch-verification of multiple blocks to reduce the average verification cost per block. The proposed scheme is secure against forgery attacks under the discrete log assumption. The proposed scheme incurs the smallest per-block communication overhead and provides the fastest block signing rates compared to existing stream authentication schemes of Wong et al. [82] and Rohatgi et al. [71] that also provide the ability to independently verify each block in the stream.

6.2 Future research directions

Future research work on topics covered by this dissertation can proceed along the following three directions: (1) Studying the feasibility of building additional forms of signatures using trapdoor hash functions; (2) Further improving efficiency of proposed trapdoor hash-based signature schemes; (3) Building practice-oriented security proofs for the proposed signature schemes.

6.2.1 Building additional forms of signatures using trapdoor hash functions

The proposed trapdoor hash-based authentication techniques are all based on the idea of replacing traditional hash functions in signature schemes with trapdoor hash functions and authenticating new messages by finding a collision with a previously signed message using a trapdoor key. This general idea can be applied to develop additional forms of digital signature schemes.

We are currently in the process of developing an aggregate signature scheme using trapdoor hash functions. The paradigm of signature aggregation allows combining multiple signatures into a single *condensed* signature, whose verification simultaneously establishes the validity of all component signatures. Depending on the type of message each participant signs, condensed signature schemes can be classified into multisignature and aggregate signature schemes. An aggregate signature scheme involves compression of n signatures $\sigma_1, \ldots, \sigma_n$, where each σ_i is a signature on a *distinct* message m_i under public key PK_i, into a single aggregate signature σ on messages $m_1, \ldots, m_n$ that can be verified under public keys $PK_1, \ldots, PK_n$. On the other hand, a multisignature is an aggregate

signature generated on a common message. Both multisignature and aggregate signature schemes can be constructed in a sequential [54] or non-sequential manner [14]. Condensed signatures offer bandwidth and storage savings, and are usually more efficient to verify compared to verifying all component signatures individually. Due to these advantages, condensed signatures have found numerous applications like, securing multicast feedback acknowledgement [19, 22], securing ad hoc routing [21, 42], authenticating delegates in mobile code systems [75], securing path-vector routing protocols [23, 41, 84] and building efficient certificate chains [54].

Assume a trapdoor hashing scheme that allows multiple entities to compute collisions with a given hash value. Intuitively, the method for producing trapdoor hash based aggregate signatures exploits the general practice of generating signatures on hashes of messages (rather than on the message itself). By replacing traditional hash functions with trapdoor hash functions in the signature generation process, each signer participating in the aggregate signature scheme can find collisions between hashes of different messages. Thus, by using trapdoor collisions among the different messages that each participant signs, each signer can produce signatures on a common hash value that can be combined using a multisignature construction technique to produce the resulting aggregate signature. This is because, as noted earlier, multisignatures are essentially aggregate signatures with the restriction that each signer signs the same message, or rather, the same hash value. This method for constructing aggregate signatures represents a significant departure from traditional aggregate signature construction techniques and can provide researchers with new opportunities to built novel aggregate signature schemes that can take advantage of various existing multisignature construction techniques to achieve the desired efficiency and scalability needs. For instance, the multisignature by Micali et al. [61] outperforms all known aggregate signatures in terms of computation overhead for signature generation, aggregation and verification.

6.2.2 Further improving efficiency of proposed trapdoor hash-based signature schemes

The schemes proposed in this dissertation demonstrate improved performance over existing schemes in the literature. The discrete log-based instantiations of the proposed generic techniques serve as the first step in building efficient authentications schemes. The proposed generic techniques allow the use of any type of digital signature and trapdoor hashing schemes provided that the security properties of the constituent schemes meet the requirements described in this dissertation. Future implementations of the proposed

generic techniques can use primitives from various cryptosystems and new constructions of signature and trapdoor hashing schemes to further enhance the efficiency of the proposed trapdoor hash-based authentication schemes.

One technique to further improve the performance of the proposed schemes is through the use of linear feedback shift register (LFSR)-based cryptosystems to develop new instantiations of the proposed techniques. An nth order LFSR is an electronic switching circuit with n storage units regulated by a clock that causes the content of each storage unit to shift to the next unit in line with each clock pulse. Traditionally, LFSRs have been used to generate key streams in stream ciphers. More recently, Gong et al. [35] and Lenstra et al. [50] have proposed the use of LFSR to build public key cryptosystems. LFSR-based cryptosystems use reduced representation of finite field elements and extremely fast sequence operations to provide substantial savings of communication and computational overhead for a desired level of security. For instance, the XTR cryptosystem [50] uses 340-bit keys to provide security equivalent to a DL-based cryptosystem that uses 1024-bit keys. Moreover, sequence operations in XTR is three times faster than modular exponentiations in the DL-based cryptosystem [22, 21, 23].

We have used LFSR-based cryptosystems in the past to build efficient and scalable multisignatures [22, 21], aggregate signatures [23], blind signatures [20] and proxy signatures [25]. Future implementations of the proposed techniques can use LFSR-based cryptosystems to achieve performance improvements over the discrete log-based instantiations presented in this dissertation.

6.2.3 Building practice-oriented security proofs for the proposed signature schemes

In this dissertation we use the complexity theoretic framework to formalize the security proofs of the proposed authentication schemes. The proposed security proofs use probabilistic polynomial time adversaries, polynomial time transformations and show that the adversary succeeds in breaking the proposed scheme with negligible probability. Although such proofs form a good baseline to demonstrate the security of our schemes, their impact in practice is limited. Practitioners can better judge the security of a scheme when provided with data like the number of cycles of adversarial computations a scheme can withstand, minimum key size (or parameter size) needed to withstand attacks, and so forth.

Future work on the proposed schemes can involve *refining* the security proofs by taking a practice-oriented approach [9] where one can attempt to present quantitative results regarding the security of the proposed authentication schemes. Future security proofs

can place quantitative bounds on the success probability of an adversary in breaking the proposed schemes. The success probability can be expressed in terms of quantifiable values such as the number of oracle queries made by the adversary, the amount of time an adversary computes, the key length (or the parameter length).

Bibliography

[1] Gail-Joon Ahn, Badrinath Mohan, and Seng-Phil Hong. Towards secure information sharing using role-based delegation. *Journal of Network and Computer Applications*, 30(1):42–59, 2007.

[2] Akamai. Akamai information security management system overview: Securing the cloud. White Paper. Available from `http://www.akamai.com/security`, Accessed Jan 4, 2010.

[3] Giuseppe Ateniese and Breno de Medeiros. Identity-based chameleon hash and applications. In Ari Juels, editor, *Proceedings of FC, 8th International Conference on Financial Cryptography*, volume 3110 of *LNCS*, pages 164–180. Springer, 2004.

[4] Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In Carlo Blundo and Stelvio Cimato, editors, *Proceedings of SCN, 4th International Conference on Security in Communication Networks*, volume 3352 of *LNCS*, pages 165–179. Springer, 2004.

[5] Vijayalakshmi Atluri and Janice Warner. Supporting conditional delegation in secure workflow management systems. In Elena Ferrari and Gail-Joon Ahn, editors, *Proceedings of SACMAT, 10th ACM Symposium on Access Control Models and Technologies, Stockholm, Sweden, June 1-3*, pages 49–58. ACM, 2005.

[6] Amit K. Awasthi and Sunder Lal. Proxy blind signature scheme. *Transactions on Cryptology.*, 2(1):5–1, 2005.

[7] Ezedin Barka and Ravi S. Sandhu. Framework for role-based delegation models. In *Proceedings of ACSAC, 16th Annual Computer Security Applications Conference*, page 168. IEEE Computer Society, 2000.

[8] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. On the selection of pairing-friendly groups. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Proceedings*

of SAC, Tenth Annual International Workshop on Selected Areas in Cryptography, Revised Papers, volume 3006 of *LNCS*, pages 17–25. Springer, 2003.

[9] Mihir Bellare. Practice-oriented provable-security. In Eiji Okamoto, George I. Davida, and Masahiro Mambo, editors, *Proceedings of ISW, First International Information Security Workshop, Tatsunokuchi, Japan, September 17-19, 1997*, volume 1396 of *LNCS*, pages 221–231. Springer, 1998.

[10] Mihir Bellare. Practice-oriented provable security. In Ivan Damgård, editor, *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, July 1998*, volume 1561 of *LNCS*, pages 1–15. Springer, 1999.

[11] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of CCS, First ACM conference on Computer and communications security*, pages 62–73. ACM Press, 1993.

[12] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs #1. In Hugo Krawczyk, editor, *Proceedings of CRYPTO, Advances in Cryptology, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998*, volume 1462 of *LNCS*, pages 1–12. Springer, 1998.

[13] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. Cryptology ePrint Archive, Report 2003/096, 2008. `http://eprint.iacr.org/2003/096`.

[14] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Proceedings of EUROCRYPT: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 2656 of *LNCS*, pages 416–432. Springer, 2003.

[15] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.

[16] Peter Bright. Google, Microsoft distribute malware after domain name trickery. Ars Technica, December 13 2010. Available from `http://arstechnica.com/security/news/2010/12/google-microsoft-distribute-malware-after-domain-name-trickery.ars`, Accessed December 21, 2010.

[17] Randy Butler, Von Welch, Douglas Engert, Ian T. Foster, Steven Tuecke, John Volmer, and Carl Kesselman. A national-scale authentication infrastructur. *IEEE Computer*, 33(12):60–66, 2000.

[18] Srdjan Capkun, Mario Cagalj, Ram Kumar Rengaswamy, Ilias Tsigkogiannis, Jean-Pierre Hubaux, and Mani B. Srivastava. Integrity codes: Message integrity protection and authentication over insecure channels. *IEEE Transactions on Dependable and Secure Computing*, 5(4):208–223, 2008.

[19] Claude Castelluccia, Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Secure acknowledgment aggregation and multisignatures with limited robustness. *Computer Networks*, 50(10):1639–1652, 2006.

[20] Saikat Chakrabarti, Santosh Chandrasekhar, Kenneth L. Calvert, and Mukesh Singhal. Efficient blind signatures for accountability. In *Proceedings of NPSec: The Third Workshop on Secure Network Protocols, Beijing, China*, October 2007.

[21] Saikat Chakrabarti, Santosh Chandrasekhar, Mukesh Singhal, and Kenneth L. Calvert. Authenticating DSR using a novel multisignature scheme based on cubic LFSR sequences. In Frank Stajano, Catherine Meadows, and Srdjan Capkun, editors, *Proceedings of ESAS: The Fourth European Workshop on Security and Privacy in Ad hoc and Sensor Networks*, volume 4572 of *LNCS*, pages 156–171. Springer, 2007.

[22] Saikat Chakrabarti, Santosh Chandrasekhar, Mukesh Singhal, and Kenneth L. Calvert. Authenticating feedback in multicast applications using a novel multisignature scheme based on cubic LFSR sequences. *AINAW: 21st International Conference on Advanced Information Networking and Applications Workshops*, 1:607–613, 2007.

[23] Saikat Chakrabarti, Santosh Chandrasekhar, Mukesh Singhal, and Kenneth L. Calvert. An efficient and scalable quasi-aggregate signature scheme based on lfsr sequences. *IEEE Transactions on Parallel and Distributed Systems*, 20(7):1059–1072, 2009.

[24] Santosh Chandrasekhar, Saikat Chakrabarti, and Mukesh Singhal. Efficient proxy signatures for ubiquitous computing. In *Proceedings of SUTC, IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, Taichung, Taiwan*, June 2008.

[25] Santosh Chandrasekhar, Saikat Chakrabarti, and Mukesh Singhal. Efficient proxy signatures for ubiquitous computing. *Proceedings of SUTC, IEEE International*

Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, 0:106–113, 2008.

[26] Xiaofeng Chen, Fangguo Zhang, and Kwangjo Kim. Chameleon hashing without key exposure. In Kan Zhang and Yuliang Zheng, editors, *Proceedings of ISC, Information Security, 7th International Conference, Palo Alto, CA, USA, September 27-29*, volume 3225 of *LNCS*, pages 87–98. Springer, 2004.

[27] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[28] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Strong key-insulated signature schemes. In Yvo Desmedt, editor, *Proceedings of PKC, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003*, volume 2567 of *LNCS*, pages 130–144. Springer, 2003.

[29] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital schemes. In Gilles Brassard, editor, *Proceedings of CRYPTO, 9th Annual International Cryptology Conference*, volume 435 of *LNCS*, pages 263–275. Springer, 1989.

[30] Federal Information Processing Standards (FIPS). The keyed-hash message authentication code (HMAC), March 2002.

[31] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *Proceedings of CCS, Fifth ACM Conference on Computer and Communications Security*, pages 83–92. ACM, 1998.

[32] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[33] Rosario Gennaro and Pankaj Rohatgi. How to sign digital streams. In *Proceedings of CRYPTO, 17th Annual International Cryptology Conference*, volume 1294 of *LNCS*, pages 180–197. Springer, 1997.

[34] Philippe Golle and Nagendra Modadugu. Authenticating streamed data in the presence of random packet loss. In *Proceedings of NDSS, Network and Distributed System Security Symposium*. The Internet Society, 2001.

[35] Guang Gong and Lein Harn. Public-key cryptosystems based on cubic finite field extensions. *IEEE Transactions on Information Theory*, 45(7):2601–2605, 1999.

[36] Antone Gonsalves. YouTube confirms Justin Bieber hack attack. InformationWeek, July 1 2010. Available from `http://www.informationweek.com/news/security/attacks/showArticle.jhtml?articleID=225702490`, Accessed Dec 21, 2010.

[37] Lein Harn, Wen-Jung Hsin, and Changlu Lin. Efficient on-line/off-line signature schemes based on multiple-collision trapdoor hash families. *The Computer Journal*, 53(9):1478–1484, 2010.

[38] Patrick Horster, Holger Petersen, and Markus Michels. Meta-elgamal signature schemes. In *Proceedings of CCS: Second ACM Conference on Computer and Communications Security*, pages 96–107. ACM Press, 1994.

[39] Xinyi Huang, Willy Susilo, Yi Mu, and Wei Wu. Proxy signature without random oracles. In Jiannong Cao, Ivan Stojmenovic, Xiaohua Jia, and Sajal K. Das, editors, *Proceedings of MSN, Second International Conference on Mobile Ad-hoc and Sensor Networks*, volume 4325 of *LNCS*, pages 473–484. Springer, 2006.

[40] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM (CACM)*, 44(4):35–41, 2001.

[41] Stephen T. Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure border gateway protocol (S-BGP) - real world performance and deployment issues. In *Proceedings of NDSS: the Network and Distributed System Security Symposium, San Diego, California, USA*. The Internet Society, 2000.

[42] Jihye Kim and Gene Tsudik. SRDP: Securing route discovery in DSR. In *Proceedings of MobiQuitous: Second Annual International Conference on Mobile and Ubiquitous Systems, 17-21 July, San Diego, CA, USA*, pages 247–260. IEEE Computer Society, 2005.

[43] Seungjoo Kim, Sangjoon Park, and Dongho Won. Proxy signatures, revisited. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *Proceedings of ICICS, First International Conference on Information and Communication Security*, volume 1334 of *LNCS*, pages 223–232. Springer, 1997.

[44] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Proceedings of NDSS, Network and Distributed System Security Symposium*. The Internet Society, 2000.

[45] Sunder Lal and Amit K. Awasthi. A scheme for obtaining a warrant message from the digital proxy signatures. Cryptology ePrint Archive, Report 2003/073, 2003. `http://eprint.iacr.org/2003/073`.

[46] Byoungcheon Lee, Heesun Kim, and Kwangjo Kim. Secure mobile agent using strong non-designated proxy signature. In Vijay Varadharajan and Yi Mu, editors, *Proceedings of ACISP, 6th Australasian Conference on Information Security and Privacy*, volume 2119 of *LNCS*, page 474. Springer, 2001.

[47] Byoungcheon Lee, Heesun Kim, and Kwangjo Kim. Strong proxy signature and its application. In *Proceedings of SCIS, Symposium on Cryptography and Information Security Osio, Japan*, volume 2/2, pages 603–608, 2001.

[48] Jung-Yeun Lee, Jung Hee Cheon, and Seungjoo Kim. An analysis of proxy signatures: Is a secure channel necessary? In Marc Joye, editor, *Proceedings of CT-RSA, Cryptographers' Track at the RSA Conference*, volume 2612 of *LNCS*, pages 68–79. Springer, 2003.

[49] Jussipekka Leiwo, Christoph Hänle, Philip Homburg, and Andrew S. Tanenbaum. Disallowing unauthorized state changes of distributed shared objects. In Sihan Qing and Jan H. P. Eloff, editors, *Information Security for Global Information Infrastructures, IFIP TC11 Fifteenth Annual Working Conference on Information Security*, volume 175 of *IFIP Conference Proceedings*, pages 381–390. Kluwer, 2000.

[50] Arjen K. Lenstra and Eric R. Verheul. The XTR Public Key System. In Mihir Bellare, editor, *Proceedings of CRYPTO: 12th Annual International Cryptology Conference on Advances in Cryptology*, volume 1880 of *LNCS*, pages 1–19. Springer, 2000.

[51] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In Yvo Desmedt, editor, *Proceedings of CRYPTO, Advances in Cryptology, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25*, volume 839 of *LNCS*, pages 95–107. Springer, 1994.

[52] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *Proceedings of EUROCRYPT, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 4004 of *LNCS*, pages 465–485. Springer, 2006.

[53] Bjoern M. Luettmann and Adam C. Bender. Man-in-the-middle attacks on auto-updating software. *Bell Labs Technical Journal*, 12(3):131138, 2007.

[54] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *Proceedings of EUROCRYPT: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3027 of *LNCS*, pages 74–90. Springer, 2004.

[55] Anna Lysyanskaya, Roberto Tamassia, and Nikos Triandopoulos. Authenticated error-correcting codes with applications to multicast authentication. *ACM Transactions on Information and System Security*, 13(2), 2010.

[56] Tal Malkin, Satoshi Obana, and Moti Yung. The hierarchy of key evolving signatures and a characterization of proxy signatures. In Christian Cachin and Jan Camenisch, editors, *Proceedings of Eurocrypt, International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3027 of *LNCS*, pages 306–322. Springer, 2004.

[57] Masahiro Mambo, Keisuke Usuda, and Eiji Okamoto. Proxy signatures for delegating signing operation. In *Proceedings of CCS, Third ACM Conference on Computer and Communications Security*, pages 48–57. ACM Press, 1996.

[58] Ueli M. Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *Proceedings of, Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005*, volume 3796 of *LNCS*, pages 1–12. Springer, 2005.

[59] Manish Mehta and Lein Harn. Efficient one-time proxy signatures. *IEE Proceedings Communications*, 152(2):129–133, 2005.

[60] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography (Discrete Mathematics and Its Applications)*. CRC Press, first edition, December 16 1996.

[61] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In *Proceedings of CCS: Eighth ACM conference on Computer and Communications Security*, pages 245–254. ACM Press, 2001.

[62] Dejan Milojicic. Trend wars - mobile agent applications. *IEEE Concurrency*, 7(3):80–90, 1999.

[63] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Proceedings of CRYPTO, Advances in Cryptolog, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, 2002.

[64] Kaisa Nyberg and Rainer A. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. In Alfredo De Santis, editor, *Proceedings of EUROCRYPT: Workshop on the Theory and Application of Cryptographic Techniques*, volume 950 of *LNCS*, pages 182–193. Springer, 1994.

[65] Takeshi Okamoto, Mitsuru Tada, and Eiji Okamoto. Extended proxy signatures for smart cards. In Masahiro Mambo and Yuliang Zheng, editors, *Proceedings of ISW, Second International Workshop on Information Security*, volume 1729 of *LNCS*, pages 247–258. Springer, 1999.

[66] Jung Min Park, Edwin K. P. Chong, and Howard Jay Siegel. Efficient multicast stream authentication using erasure codes. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):258–285, 2003.

[67] Adrian Perrig. The BiBa one-time signature and broadcast authentication protocol. In *ACM Conference on Computer and Communications Security*, pages 28–37, 2001.

[68] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Xiaodong Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, 2000.

[69] Holger Petersen and Patrick Horster. Self-certified keys – concepts and applications. In *Proceedings of CMS, 3rd International Conference on Communications and Multimedia Security*, pages 102–116. Chapman & Hall, 1997.

[70] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Proceedings of EUROCRYPT, Advances in Cryptology, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996*, volume 1070 of *LNCS*, pages 387–398. Springer, 1996.

[71] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *ACM Conference on Computer and Communications Security*, pages 93–100, 1999.

[72] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[73] Jacob C. N. Schuldt, Kanta Matsuura, and Kenneth G. Paterson. Proxy signatures secure against proxy key exposure. In Ronald Cramer, editor, *Proceedings of PKC, 11th International Workshop on Practice and Theory in Public-Key Cryptography*, volume 4939 of *LNCS*, pages 141–161. Springer, 2008.

[74] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *Proceedings of CRYPTO, Advances in Cryptology, 21st Annual International Cryptology Conference*, volume 2139 of *LNCS*, pages 355–367. Springer, 2001.

[75] Shiuh-Pyng Shieh, Chern-Tang Lin, Wei-Bon Yang, and Hung-Min Sun. Digital multisignature schemes for authenticating delegates in mobile code systems. *IEEE Transactions on Vehicular Technology*, 49(4):1464–1473, July 2000.

[76] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Proceedings of EUROCRYPT, Advances in Cryptology, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997*, volume 1233 of *LNCS*, pages 256–266. Springer, 1997.

[77] Federal Information Processing Standards. Digital signature standard (DSS), June 2009.

[78] Hung-Min Sun. An efficient nonrepudiable threshold proxy signature scheme with known signers. *Computer Communications*, 22(8):717–722, 1999.

[79] Hung-Min Sun and Bin-Tsan Hsieh. Remarks on two nonrepudiable proxy signature schemes. In *Proceedings of 9th National Conference on Information Security*, pages 241–246, 1999.

[80] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Computer Communication Review*, 39(1):50–55, 2009.

[81] Jon Weissman and Siddharth Ramakrishnan. Using proxies to accelerate cloud applications. In *Proceedings of HotCloud, Workshop on Hot Topics in Cloud Computing in conjunction with the 2009 USENIX Annual Technical Conference, June 14-19, San Diego, CA, USA*. USENIX, 2009.

[82] Chung Kei Wong and Simon S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Transactions on Networking*, 7(4):502–513, 1999.

[83] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. An efficient signature scheme from bilinear pairings and its applications. In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *Proceedings of PKC, 7th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2947 of *LNCS*, pages 277–290. Springer, 2004.

[84] Meiyuan Zhao, Sean W. Smith, and David M. Nicol. Aggregated path authentication for efficient BGP security. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *Proceedings of CCS '05: 12th ACM Conference on Computer and Communications Security, November 7-11, Alexandria, VA, USA*, pages 128–138. ACM Press, 2005.

VITA

Name: Santosh Chandrasekhar

Date and Place of Birth: 23^{rd} March 1981, Chennai, India.

Education: B.E., Computer Science and Engineering
Sri Chandrasekharendra Saraswathi Viswa Maha Vidyalaya
India, 2002

Employment History:

1. Jan 2009—Present: Research Assistant, University of Kentucky
2. Aug 2008—Dec 2008: Teaching Assistant, University of Kentucky
3. May 2004—Aug 2008: Research Assistant, University of Kentucky
4. Jan 2004—May 2004: Graduate-level Course Grader, University of Kentucky
5. Aug 2003—Dec 2003: Teaching Assistant, University of Kentucky
6. May 2003—Aug 2003: Research Assistant, University of Kentucky
7. Jan 2003—May 2003: Graduate-level Course Grader, University of Kentucky
8. Jun 2001—May 2002: Intern, Harita Techserve Ltd.

Honors:

- Verizon Communications Graduate Fellowship, University of Kentucky, 2008—2009
- Kentucky Graduate Scholarship, University of Kentucky, 2002—2008
- One of two departmental nominee for 2010 University of Kentucky Presidential Fellowship Award
- Ranked 2^{nd} among 250 B.E. computer science students in graduating class of 2002

Selected Publications:

1. Saikat Chakrabarti, Santosh Chandrasekhar, Mukesh Singhal and Kenneth L. Calvert, "Authenticating Feedback in Multicast Applications Using a Novel Multisignature Scheme Based on Cubic LFSR Sequences", in *Proceedings of SSNDS: The Third*

IEEE International Symposium on Security in Networks and Distributed Systems, in conjunction with AINA, Niagara Falls, Canada, May 21-23, 2007, IEEE Computer Society, vol. 1, 2007, pp. 607-613

2. Saikat Chakrabarti, Santosh Chandrasekhar, Mukesh Singhal and Kenneth L. Calvert, "Authenticating DSR using a Novel Multisignature Scheme based on Cubic LFSR Sequences", in *Proceedings of ESAS: The Fourth European Workshop on Security and Privacy in Ad hoc and Sensor Networks*, Cambridge, United Kingdom, July 2-3, ser. LNCS, vol. 4572, Springer, 2007, pp. 156-171

3. Saikat Chakrabarti, Santosh Chandrasekhar, Kenneth L. Calvert and Mukesh Singhal, "Efficient Blind Signatures for Accountability", in *Proceedings of NPSec: The Third Workshop on Secure Network Protocols, in conjunction with the 15th IEEE International Conference on Network Protocols (ICNP)*, Beijing, China, October 16-19, 2007, IEEE Xplore 2.0, pp 01-06

4. Santosh Chandrasekhar, Saikat Chakrabarti and Mukesh Singhal, "Efficient Proxy Signatures For Ubiquitous Computing", in *Proceedings of SUTC: IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, Taichung, Taiwan, June 11-13, 2008

5. Saikat Chakrabarti, Santosh Chandrasekhar, Mukesh Singhal and Kenneth L. Calvert, "An Efficient and Scalable Quasi-Aggregate Signature Scheme Based on LFSR Sequences", in *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Vol. 20, Issue 7, July 2009, pp. 1059-1072

6. Saikat Chakrabarti, Santosh Chandrasekhar and Mukesh Singhal, "An Escrow-less Identity-based Group Key Agreement Protocol for Dynamic Peer Groups", in *International Journal of Security and Networks (IJSN)*, Vol. 4, No. 3, 2009, pp. 171-188

7. Santosh Chandrasekhar, Saikat Chakrabarti, Mukesh Singhal and Kenneth L. Calvert, "Efficient proxy signatures based on trapdoor hash functions", in *IET Information Security, Special Issue: Selected papers on multi-agent and distributed information security*, Vol. 4, No. 4, December 2010, pp. 322-332