One-Way Signature Chaining - A New Paradigm For Group Cryptosystems And E-Commerce

Amitabh Saxena and Ben Soh
Dept. of Computer Science and Computer Engineering
La Trobe University, Bundoora, VIC, Australia 3086

November 30, 2006

Abstract

In this paper, we describe a new cryptographic primitive called *(One-Way) Signature Chaining.* Signature chaining is essentially a method of generating a chain of signatures on the same message by different users. Each signature acts as a "link" of the chain. The *one-way*-ness implies that the chaining process is one-way in the sense that more links can be easily added to the chain. However, it is computationally infeasible to remove any intermediate links without removing all the links. The signatures so created are called chain signatures (CS). We give precise definitions of chain signatures and discuss some applications in trust transfer. We then present a practical construction of a CS scheme that is secure (in the random oracle model) under the Computational Diffie-Hellman (CDH) assumption in bilinear maps.

1 Introduction

Over recent years, a lot of research in e-commerce systems has been on the problem of *trust transfer*. Roughly speaking, trust transfer is the act of transferring the trust placed on the original user (the *trusted*) to a proxy user (the *trustee*) such that some other user (the *truster*) can delegate the same responsibilities to the trustee that he would have delegated to the trusted in some *trust context* (for instance, a electronic transaction).

In this paper, we will attempt to give a formal treatment of this trust transfer using the notion of **Chain signatures** (CS). Chain signatures are similar to proxy signatures, where the original signer delegates signing power to a proxy signer [1]. In addition, chain signatures ensure that the hierarchy of the delegation is preserved and cannot be tampered with. The crucial difference is that chain signatures are completely non-interactive and stateless - the signer can be completely oblivious of the receiver's identity. The intriguing part of chain signatures is that despite this anonymity, they provide sufficient guarantee of the path from which this delegation was actually propagated. As an application of CS, consider wireless and ad-hoc sensor networks, where routing information often needs to be transmitted without prior knowledge of the topology.

Since in our model, the liability of authenticating the path to the next level always rests with the receiver (and never the sender), we coin the term *one-way chaining* to indicate this asymmetry. In many ways, CS are similar to transitive signatures [2] in that they allow trust to be transferred between multiple entities. The difference is that transitive signatures attempt to hide the intermediate nodes of trust transfer, while CS try to ensure that intermediate nodes cannot be removed. Intuitively, chain signatures can be considered as a combination of Verifiably Encrypted Signatures (VES) [3] and sequential aggregate signatures [4, 5].

The rest of this paper is organized as follows. We give the motivation for chain signatures in Section 2. We then formalize the intuition of chain signatures in Section 3.2. Finally, in Section 4 we present the scheme and prove its security in Section 4.3. We then discuss some applications of chain signatures in section 5.

2 Motivation

To see the motivation behind chain signatures consider a scenario with three users Alice, Bob and Carol. Alice signs some message m and sends the signature σ_A to Bob, after which she is not available for interaction. Now Bob wants to convince Carol that Alice indeed signed the message m. However if Carol later wants to convince a third party (using Bob's proof) the statement "**Alice signed the message** m" then she must only be able to do so by proving that "**Bob knows about it too**".

2.1 Intuition Behind Chain Signatures

Assume that users A, B compute signatures σ_A, σ_B (on the same message) using private keys SK_A, SK_B respectively. Define the following properties:

1. **Aggregation**: Given signatures σ_A, σ_B it is easy to compute a combined signature $\sigma_{\{A,B\}}$ that can be verified using public keys PK_A, PK_B.

2. **Delete Protection**: Given $\{\sigma_{\{A,B\}}, \mathrm{PK}_A, \mathrm{PK}_B\}$, it must be infeasible to compute σ_A or σ_B

3. **Strong Delete Protection**: This is a stronger variant of the previous property.

   - Given $\{\sigma_{\{A,B\}}, \mathrm{PK}_A, \mathrm{PK}_B, \mathrm{SK}_A\}$, it must be infeasible to compute σ_B.
   - Given $\{\sigma_{\{A,B\}}, \mathrm{PK}_A, \mathrm{PK}_B, \mathrm{SK}_B\}$, it must be infeasible to compute σ_A.

A signature scheme that satisfies the aggregation and delete protection conditions is called a *Chain Signature* (CS) scheme. A CS scheme that additionally satisfies the strong delete protection condition is called a *Strong Chain Signature* (SCS) scheme. The above idea can be extended to an arbitrary number of users.

It is fairly trivial to extend the previous argument to arbitrary number of distinct users. Assume that users $1, 2, \ldots n$ compute signatures $\sigma_1, \sigma_2, \ldots \sigma_n$ (on the same message) using private keys SK_1, $\mathrm{SK}_2, \ldots, \mathrm{SK}_n$ respectively. We can then similarly define:

1. **Aggregation**: Given signatures $\sigma_1, \sigma_2, \ldots \sigma_n$, it is easy to compute a combined signature $\sigma_{\{1,2,\ldots,n\}}$ that can be verified using public keys PK_1,$\mathrm{PK}_2, \ldots, \mathrm{PK}_n$.

2. **Delete Protection**: Given $\{\sigma_{\{1,2,\ldots,n\}}$,$\mathrm{PK}_1$,$\mathrm{PK}_2, \ldots \mathrm{PK}_n\}$, it must be infeasible to compute σ_α for any $\alpha \subsetneq \{1, 2 \ldots n\}$

3. **Strong Delete Protection**: Given $\{\sigma_{\{1,2,\ldots,n\}}$,$\mathrm{PK}_1$,$\mathrm{PK}_2, \ldots, \mathrm{PK}_n$,$\mathrm{SK}_{\beta_1}$,$\mathrm{SK}_{\beta_2}, \ldots, \mathrm{SK}_{\beta_i}\}$ for $i < n$ and $\{\beta_1, \beta_2, \ldots \beta_i\} \subsetneq \{1, 2, \ldots\}$, it must be infeasible to compute σ_α for $\alpha = \{1, 2, \ldots n\} \backslash \{\beta_1, \beta_2, \ldots \beta_i\}$.

Although in the above (informal) description we assumed that the combined signature is "unordered", in our formal definition we will also take into account the order in which the users contribute. Our model of CS will not provide "strong delete protection". However, it will provide "delete protection".

2.2 Physical Analogue Of Chain Signatures

Chain signatures can be intuitively visualized by considering a box with a link and a set of "intermediate" links with an *asymmetric* combination lock, as shown in Figure 1. In an asymmetric combination lock, the opening combination is different from the closing combination and cannot be derived from it. The initiator sends the box along with several open links and their closing combination(s) (but not the tags). The opening combination(s) are kept secret. Each user may then add a private tag to the "message", which is the equivalent of authentication. A signature is considered valid if there are no "loops" in the chain, each link has a tag, and all the tags are unique.

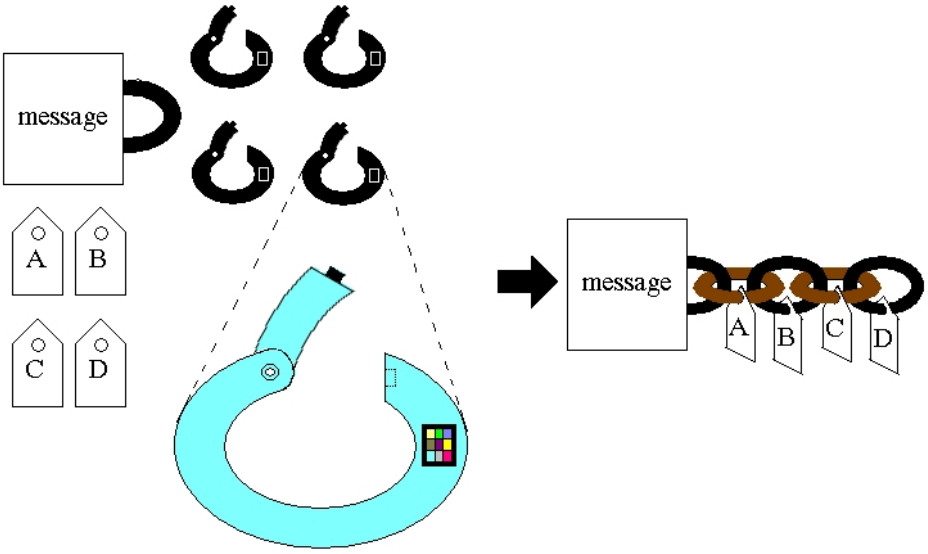


Figure 1: Physical analogue of a chain signature.

3 Formal Definition - Chain Signatures

In this section, we will formalize the above intuition of chain signatures. Since chain signatures inherently deal with ordered elements (i.e. the public keys), we first develop some notation to deal with ordered elements, which we call sequences.

1. A *sequence* is similar to a set except that the order of its elements matters. We require that the elements of a sequence must be distinct. The elements of a sequence are written in order and enclosed with $\langle , \rangle$ symbols. For instance, $\langle y_1, y_2, y_3 \rangle$.

2. The symbol θ denotes the empty sequence with zero elements. The symbol ϵ denotes the empty string of zero length.

3. Let $L_a = \langle y_1, y_2, \ldots, y_k \rangle$ be some non-empty sequence. For any other sequence L_b, we say that $L_b \prec L_a$ if and only if $L_b = \langle y_1, y_2, \ldots, y_i \rangle$ and $0 \leq i \leq k$.

4. We say that two sequences $\{L_a, L_b\}$ **overlap** if there exists a non-empty sequence L' such that $L' \prec L_a$ and $L' \prec L_b$. For instance, $\{\langle y_1, y_2 \rangle, \langle y_1 \rangle\}$ overlap, while $\{\langle y_1, y_2 \rangle, \langle y_2 \rangle\}$ do not.

5. For any two sequences L_a, L_b, the symbol $L_a \cup L_b$ denotes the **set** of elements that belong to at least one of $\{L_a, L_b\}$. Similarly $L_a \cap L_b$ denotes the **set** of elements that belong to both L_a and L_b. We denote by $L_a \odot L_b$ to be the **set** of elements from the largest sequence L' such that $L' \prec L_a$ and $L' \prec L_b$. Clearly, for two overlapping sequences $\{L_a, L_b\}$, we have that $L_a \odot L_b \neq \emptyset$.

3.1 Algorithms

A chain signature scheme is defined using three PPT algorithms KeyGen, ChainSign, ChainVerify as follows. (It is more convenient to describe ChainVerify before ChainSign.)

KeyGen (Key Generation) This *randomized* algorithm takes as input a security parameter τ and outputs a randomly selected key-pair (x, y) such that x is the private key and y is the public key. We say that $(x_i, y_i) \overset{R}{\leftarrow}$ KeyGen on the i^{th} run of this algorithm.

ChainVerify (Verification) This algorithm takes as input a tuple (m, σ_i, L_i). Here $L_i = \langle y_1, y_2, \ldots, y_i \rangle$ is some sequence of i public keys and the pair (σ_i, L_i) is a purported chain signature on message m. The algorithm works as follows:

1. If $L_i = \theta$ and $\sigma_i = \epsilon$ the algorithm outputs VALID and terminates.
2. If $L_i = \theta$ and $\sigma_i \neq \epsilon$ the algorithm outputs INVALID and terminates.
3. If this step is executed then $L_i \neq \theta$. The algorithm invokes a deterministic poly-time procedure after which it outputs either VALID or INVALID and terminates.

ChainSign (Signing) The ChainSign algorithm takes as input a tuple $(x_i, y_i, m, \sigma_j, L_j)$. Here (x_i, y_i) is a valid private-public key-pair (generated using the KeyGen algorithm), the pair (σ_j, L_j) is a purported chain signature on message m, and $L_j = \langle y_1, y_2, \ldots, y_j \rangle$ is some sequence of j public keys such that $y_i \notin \{y_1, y_2, \ldots, y_j\}$. The algorithm works as follows:

1. If any of the input conditions (as described above) are violated, the algorithm outputs ERROR and terminates.
2. The algorithm invokes ChainVerify with (m, σ_j, L_j) as input (i.e. it checks whether (σ_j, L_j) is a valid chain signature on m or not). If (σ_j, L_j) is not a valid chain signature on message m, the algorithm outputs ERROR and terminates.
3. If this step is executed then no input conditions are violated and (σ_j, L_j) is a valid chain signature on m. In this case this algorithm uses the private key x_i to compute a new valid chain signature (σ_i, L_i) on message m such that $L_i = \langle y_1, y_2, \ldots, y_j, y_i \rangle$. It outputs (σ_i, L_i) and terminates.

The ChainVerify and ChainSign algorithms must satisfy the standard consistency constraint of signatures. That is, if the input (m, σ_i, L_i) to the ChainVerify is the output of the ChainSign algorithm then the ChainVerify algorithm must output VALID. Note that ChainSign can be initialized by setting $\sigma_j = \epsilon$ and $L_j = \theta$.

3.2 Security Model

We define adaptive security of chain signatures using Game 1 below. For simplicity, we assume that the adversary is not allowed to use a chosen private key. The adversary is, however, allowed to extract private keys of choice. In this respect, our model is similar to an identity based system. We call this *adaptive security under known key and chosen message attack*.

The reader should note that this is a weaker model than *adaptive security under chosen key and chosen message attack* used in the aggregate signatures of [3]. We feel, however, that our notation is more suitable in modeling the requirements of chain signatures (which are slightly different from aggregate signatures).

Game 1

1. *Setup*: The challenger $\mathcal{C}$ sets a parameter τ and gives it to the adversary $\mathcal{A}$, who then selects a game parameter n. On receiving n, $\mathcal{C}$ generates n key-pairs $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n) \overset{R}{\leftarrow}$ **KeyGen** and gives the set $Y = \{y_1, y_2, \ldots y_n\}$ of n public keys to $\mathcal{A}$. Denote by L the set of all non-empty sequences with elements from Y.

2. *Queries*: Working adaptively, the adversary $\mathcal{A}$ issues at most q_s chain sign queries and q_e (private-key) extract queries as follows:

   (a) *ChainSign*: A chain sign query i $(1 \leq i \leq q_s)$ consists of a pair $(m_{s(i)}, L_{s(i)}) \in \Sigma^* \times L$. The challenger responds with a valid **chain signature** $(\sigma_{s(i)}, L_{s(i)})$ on $m_{s(i)}$ computed using the **ChainSign** algorithm.

   (b) *Extract*: An extract query j ($1 \leq j \leq q_e$) consists of a public key $y_{e(j)}$. The challenger responds with the private key $x_{e(j)}$.

3. *Output*: Finally $\mathcal{A}$ outputs a message-**chain-signature** pair $(m_A, (\sigma_A, L_A))$.

4. *Result*: $\mathcal{A}$ wins the game if all the following conditions hold:

   (a) i. $L_A \in L$ and the **ChainVerify** algorithm accepts $(m_A, (\sigma_A, L_A))$ as valid.

   ii. No chain sign query has been previously made on the pair (m_A, L_A).

   iii. At least one private key corresponding to L_A has not been extracted.

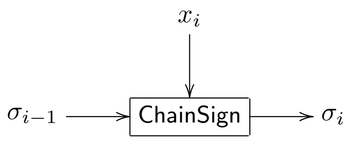
   (b) For **each** chain sign query i, if $(m_{s(i)} = m_A) \wedge (\{L_A, L_{s(i)}\}$ overlap), then there is at least one key in $(L_{s(i)} \cup L_A) \backslash (L_{s(i)} \odot L_A)$ which has not been extracted.

We use the random oracle model [6], where a hash function is implemented using a random oracle. If the adversary needs to compute a hash value it queries the random oracle, which is also simulated by the challenger. The requirement of a fair game is that the responses of the challenger (to hash queries) are indistinguishable from the responses of a random oracle.

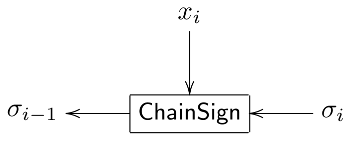
Definition 3.1. *We say that the chain signature scheme is* $(n, \tau, t, q_s, q_e, q_h, \epsilon)$*-secure under an* adaptive known-key and chosen-message attack *if, for some parameters* τ *and* n*, there is no adversary* $\mathcal{A}$ *that runs for at most time* t*; makes at most* q_s *sign queries; makes at most* q_e *extract queries; makes at most* q_h *hash queries; and wins Game 1 with probability at least* ϵ*. Otherwise, we say that* $\mathcal{A}$ $(n, \tau, t, q_s, q_e, q_h, \epsilon)$*-breaks the chain signature scheme under an adaptive known-key and chosen-message attack.*

3.3 Analysis Of Game 1

Let us analyze the Result Section of Game 1. Clearly, Part (a) rules out the cases of a trivial win. The intuition of CS is captured in Part (b). To see this, consider the illustration of the **ChainSign** algorithm at some stage i.



Since we know that **ChainSign** can be "reversed" from σ_i (using the private key x_i) to obtain σ_{i-1}, we can also consider the following figure as a valid usage of this algorithm.



Therefore, if L_A and $L_{s(i)}$ overlap for some i and $m_A = m_{s(i)}$ (we call such queries *non-trivial queries*), then we know that **ChainSign** can be "reversed" from $\sigma_{s(i)}$ and then "forwarded" to obtain σ_A using only a subset of private keys for L_A.

The security requirement of CS is that this is the **only** other way to generate σ_A without knowing all the private keys of L_A. The condition of Part (b) of the Result Section implies that if the adversary does not know at least one private key needed for this "reverse-forward" operation, then this is a valid win for the adversary.

3.4 Differences With Other Signature Schemes

In this section, we briefly demonstrate how chain signatures are different from other multi-user signature schemes such as sequential aggregate signatures [4, 5], multisignatures [7], aggregate signatures [3], and structured multisignatures [8]. We distinguish between two types of forgers for chain signatures.

Type 1 Forger This adversary either does not make any non-trivial queries or, if it makes one or more non-trivial queries then for each non-trivial query j, we have that $L_A \not\prec L_{s(j)}$. We call such a forgery an *Ordinary Forgery.*

Type 2 Forger This adversary makes one or more non-trivial queries j such that $L_A \prec L_{s(j)}$. We call such a forgery an *Extraction Forgery.*

All the above mentioned signature schemes only consider type 1 forgers, while chain signatures also consider type 2. To see this, consider the following instance of Game 1 with $n = 7$ and $Y = \{y_i | 1 \leq i \leq n\}$. The adversary outputs a valid message-chain-signature pair $(m_A, (\sigma_A, L_A))$ after making five extract queries on the keys $\{y_1, y_2, y_3, y_5, y_6\}$ and three non-trivial sign queries i (i.e, $m_A = m_{s(i)}$) such that $L_{s(i)} \odot L_A \neq \emptyset$. The sequences are (keys of extract queries have a gray box):

$$L_{s(1)} = \langle \boxed{y_1}, \boxed{y_2}, \boxed{y_3}, y_4, \boxed{y_5}, \boxed{y_6}, y_7 \rangle$$

$$L_{s(2)} = \langle \boxed{y_1}, \boxed{y_2}, \boxed{y_3}, y_4, \boxed{y_6}, y_7 \rangle$$

$$L_{s(3)} = \langle \boxed{y_1}, \boxed{y_2}, \boxed{y_3} \rangle$$

$$\boxed{L_A \; = \langle \boxed{y_1}, \boxed{y_2}, \boxed{y_3}, y_4, \boxed{y_5} \rangle}$$

Since for all the sequences $L_{s(i)}$ $(1 \leq i \leq 3)$, at least one private key needed for the reverse-forward operation (described earlier) has not been extracted, the above configuration represents a win for the adversary of Game 1. The same configuration, however, represents a loss for the adversary of a suitably adapted game (adaptive chosen-key and chosen-message attack) in all the above mentioned schemes ([4, 5, 7, 3, 8]) when we keep y_4 as the challenge public key.

4 Chain Signatures Using Bilinear Maps

In this section we give a concrete example of chain signatures using bilinear maps. First we describe the underlying primitives.

Let G_1 and G_2 be two cyclic multiplicative groups both of prime order q such that computing discrete logarithms in G_1 and G_2 is intractable. A bilinear pairing is a map $\hat{e} : G_1 \times G_1 \mapsto G_2$ that satisfies the following properties [9, 10, 3].

1. *Bilinearity*: $\hat{e}(a^x, b^y) = \hat{e}(a, b)^{xy}$ $\forall a, b \in G_1$ and $x, y \in \mathbb{Z}_q$.

2. *Non-degeneracy*: If g is a generator of G_1 then $\hat{e}(g, g)$ is a generator of G_2.

3. *Computability*: The map $\hat{e}$ is efficiently computable.

The above properties also imply:

$$\hat{e}(ab, x) = \hat{e}(a, x) \cdot \hat{e}(b, x) \; \forall a, b, x \in G_1$$
$$\hat{e}(a, xy) = \hat{e}(a, x) \cdot \hat{e}(a, y) \; \forall a, x, y \in G_1$$

In a practical implementation, G_1 is a subgroup of the (additive[1]) group of points on the elliptic curve and G_2 is the multiplicative subgroup of a finite field. The map $\hat{e}$ is derived either from the modified Weil pairing [9, 10] or the Tate pairing [11]. Typically $q \geq 2^{171}$ so that the fastest algorithm for computing discrete logarithms in G_1 (Pollard's rho method [12, p.128]) takes $\geq 2^{85}$ iterations [9].

[1] Although it is conventional to use the additive notation for the group G_1, it is more convenient in our context to use the multiplicative one.

4.1 BDH Parameter Generator

Using the idea of [9], we define a Bilinear Diffie-Hellman (BDH) parameter generator **BDH** as a randomized algorithm that takes as input $\tau \in \mathbb{N}$ and outputs $(\hat{e}, q, G_1, G_2)$, where G_1, G_2 are the descriptions of two cyclic multiplicative groups, each of prime order q such that $q \approx 2^\tau$, and $\hat{e} : G_1 \times G_1 \mapsto G_2$ is a bilinear map as defined above.

The security of chain signatures depends on the hardness of the following problem.

Computational Diffie-Hellman Problem $\mathbf{CDH}_{(g,G_1)}$: Let $g \overset{R}{\leftarrow} G_1$ be a generator of G_1. Given $g^x, g^y \in G_1$ for unknown $x, y \in \mathbb{Z}$, output $g^{xy} \in G_1$.

The Computational Diffie-Hellman Assumption (CDHA) states that the $\text{CDH}_{(g,G_1)}$ problem is intractable for any PPT adversary. This is formally stated below.

CDHA. Let $\mathcal{A}$ be an algorithm, and $\nu : \mathbb{N} \mapsto [0, 1]$ a function. We associate with any $\tau \in \mathbb{N}$ the following experiment.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{cdh}(\tau)$

$(\hat{e}, q, G_1, G_2) \overset{R}{\leftarrow} \mathbf{BDH}(\tau);\ g \overset{R}{\leftarrow} G_1 \backslash \{1\};\ \alpha, \beta \overset{R}{\leftarrow} \mathbb{Z}_q; a \leftarrow g^\alpha, b \leftarrow g^\beta$

$c \leftarrow \mathcal{A}(\hat{e}, q, G_1, G_1, g, a, b)$

If $c = g^{\alpha\beta}$ return 1 else return 0

We let

$$\mathbf{Adv}_{\mathcal{A}}^{cdh}(\tau) = \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{cdh}(\tau) = 1\right]$$

denote the advantage of $\mathcal{A}$ on input τ, the probability computed over the random choices of the inputs to $\mathcal{A}$ and the coins of $\mathcal{A}$ if any. We say that $\mathcal{A}$ has success bound ν for τ if $\mathbf{Adv}_{\mathcal{A}}^{cdh}(\tau) \leq \nu(\tau)$

Assumption CDHA: For all poly-time algorithms $\mathcal{A}$ there exists a negligible function ν and an integer $\tau' \in \mathbb{N}$, such that for all $\tau > \tau'$, $\mathcal{A}$ has success bound ν.

Definition 4.1. *We say that algorithm $\mathcal{A}$ (τ, t, ϵ)-breaks the BDH parameter generator* **BDH** *if $\mathcal{A}$ runs for time at most t, and $\mathbf{Adv}_{\mathcal{A}}^{cdh}(\tau)$ is at least ϵ.*

4.2 Chain Signature Protocol

In this scenario, n ordered distinct users $\langle 1, 2, \ldots n\rangle$, $m \in \Sigma^*$ is the message or contract.

System Parameters A Trusted Authority (TA) sets the parameter τ and generates $(\hat{e}, G_1, G_2, q) \overset{R}{\leftarrow} \mathbf{BDH}(\tau)$. Here, G_1, G_2 are descriptions of two groups each of prime order $q \approx \tau$ bits and $\hat{e} : G_1 \times G_1 \mapsto G_2$ is a bilinear mapping as defined above. The TA selects a one-way hash function $\mathcal{H} : \Sigma^* \mapsto G_1$ and picks a random generator g of G_1. The system parameters are $\langle e, q, G_1, G_2, \mathcal{H}, g\rangle$.

KeyGen Each participant i generates $x_i \overset{R}{\leftarrow} \mathbb{Z}_q^*$ as the private key. The corresponding public key is $y_i = g^{x_i} \in G_1$.

ChainSign Let $L_i = \langle y_1, y_2, \ldots y_i\rangle$ and $h_i = \mathcal{H}(m, L_i)$ for $i \geq 1$. Define $\sigma_0 = 1 \in G_1$ and define recursively

$$\sigma_i = \sigma_{i-1} h_i^{x_i} \in G_1 \text{ for } i \geq 1$$

The chain signature of user i on m is (σ_i, L_i).

ChainVerify We accept the signature (σ_i, L_i) on m as valid if the following equality holds:

$$\hat{e}(\sigma_i, g) \overset{?}{=} \prod_{j=1}^{i} \hat{e}(h_j, y_j)$$

The correctness of the verification process follows from the property of bilinear maps:

$$LHS = \hat{e}(\sigma_i, g) = \hat{e}(\prod_{j=1}^{i} h_j^{x_j}, g) = \prod_{j=1}^{i} \hat{e}(h_j, g^{x_j}) = RHS$$

4.3 Security Of The Construction

We use Definition 3.1 of Section 3.2 to prove adaptive security of the chain signature scheme. We will reuse the proof of security of the signature scheme of Boneh, Lynn and Shacham [9] (hereafter called BLS). The scheme is defined as follows.

Preliminary Setup For BLS Signatures A Trusted Authority (TA) sets a security parameter τ and generates $(\hat{e}, G_1, G_2, q) \overset{R}{\leftarrow} \mathbf{BDH}(\tau)$. Here, G_1, G_2 are descriptions of two groups each of prime order $q \approx \tau$ bits and $\hat{e} : G_1 \times G_1 \mapsto G_2$ is a bilinear mapping as defined above. The TA selects a one-way hash function $\mathcal{H} : \Sigma^* \mapsto G_1$ and picks a random generator g of G_1. The system parameters are $\langle e, q, G_1, G_2, \mathcal{H}, g \rangle$.

1. **KeyGen** Generate $x \overset{R}{\leftarrow} \mathbb{Z}_q^*$ as the private key. The public key is $y = g^x \in G_1$.
2. **Sign** To sign message m under public key y, compute $h = \mathcal{H}(m)$ and $\sigma = h^x \in G_1$. The algorithm outputs σ as a valid signature on message m under public key y.
3. **Verify** Accept the signature σ under the public key y as valid if the following holds:

$$\hat{e}(\sigma, g) \overset{?}{=} \hat{e}(\mathcal{H}(m), y)$$

Security of BLS signatures is defined using Game 2.

Game 2

1. *Setup*: The challenger sets some security parameter τ' and generates a key pair $(x, y) \overset{R}{\leftarrow}$KeyGen. It gives y, the challenge public key, along with τ' to the adversary.
2. *Queries*: Working adaptively, $\mathcal{A}$ issues at most q'_s sign queries and q'_h hash queries:
   (a) *Sign queries*: For each sign query i on distinct messages m_i for $1 \leq i \leq q'_s$, the challenger responds with a valid BLS signature $\sigma_i = \mathcal{H}(m_i)^x \in G_1$.
   (b) *Hash queries*: For each hash query j on distinct messages m_h for $1 \leq j \leq q'_h$, the challenger responds with $\mathcal{H}(m_j)$.
3. *Output*: Finally $\mathcal{A}$ outputs a BLS signature σ_A.
4. *Result*: $\mathcal{A}$ wins if σ_A is a valid signature and no sign query was issued on m_A.

Definition 4.2. *We say that the BLS scheme is $(\tau', t', q'_s, q'_h \epsilon')$-secure against existential forgery under an adaptive* ***chosen message*** *attack if for some parameter τ', there is no adversary $\mathcal{A}$ that runs for at most time t'; makes at most q'_s sign queries and q_h hash queries; and wins Game 2 with probability at least ϵ'. Otherwise, if such an adversary $\mathcal{A}$ exists, then we say that $\mathcal{A}$ $(\tau', t', q'_s, q'_h, \epsilon')$-breaks the BLS scheme.*

To prove the security of chain signatures, we will use the following result from [9].

Theorem 4.3. *([9, Theorem 3.2]) If there exists an algorithm $\mathcal{A}$ that $(\tau', t', q'_s, q'_h, \epsilon')$-breaks the BLS scheme under Definition 4.2, then then there exists another algorithm $\mathcal{B}$ that $(\tau'', t'', \epsilon'')$-breaks the BDH parameter generator* **BDH** *under Definition 4.1, where;*

$$\tau'' = \tau'; \quad t'' \leq t' + c'_{G_1}(q'_h + 2q'_s); \quad \epsilon'' \geq \frac{\epsilon'}{e(q'_s + 1)}$$

Here, c'_{G_1} is a constant that depends on G_1 and e is the base of natural logarithms.

Our security follows from Theorem 4.4 below.

Theorem 4.4. *If there exists an algorithm that $\mathcal{A}$ that $(n, \tau, t, q_s, q_e, q_h, e)$-breaks the chain signature scheme under Definition 3.1, then there exists another algorithm $\mathcal{B}$ that $(\tau', t', q_s', q_h', e')$-breaks the BLS scheme under Definition 4.2, where;*

$$\tau = \tau'; \quad t' \leq t + c_{G_1}\Big[n(q_h + 1) + q_s(n+1)\Big]; \quad q_s' \leq q_s; \quad q_h' \leq n(q_h + q_s); \quad \epsilon' \geq \frac{\epsilon}{2eq_e}$$

Here, c_{G_1} is a constant that depends on G_1 and e is the base of natural logarithms.

Proof. Let there exist an adversary $\mathcal{A}$ that $(n, \tau, t, q_s, q_e, q_h, e)$-breaks the chain signature scheme under an adaptive known key and chosen message attack. Using $\mathcal{A}$, we construct another algorithm $\mathcal{B}$ that $(\tau', t', q_s', q_h', e')$-breaks the BLS signature scheme under an adaptive chosen message attack.

Algorithm $\mathcal{B}$ simulates the adversary of Game 2 and is given a challenge public key $(g, y) = (g, g^x) \in {G_1}^2$ (for unknown x), along with a security parameter τ' by challenger $\mathcal{C}$. Its goal is to forge a valid BLS signature under y using adversary $\mathcal{A}$, that can win Game 1. Algorithm $\mathcal{B}$ simulates the challenger of Game 1 to adversary $\mathcal{A}$ as follows.

Setup. Algorithm $\mathcal{B}$ maintains a table of n entries called the K-List. Each entry i in the table is a tuple of the form $(a_i, r_i, y_i) \in \{0,1\} \times \mathbb{Z}_q^* \times G_1$ and is created as follows. First $\mathcal{B}$ generates $r_i \overset{R}{\leftarrow} \mathbb{Z}_q$ and a bit $a_i \overset{R}{\leftarrow} \{0,1\}$ using a biased coin such that $\Pr[a_i = 1] = 1/q_e$. It then computes $y_i = y^{a_i} g^{r_i} \in G_1$ and gives the set $Y = \{y_1, y_2, \ldots y_n\}$ as the challenge public keys, along with τ to $\mathcal{A}$.

Queries. To handle the queries of $\mathcal{A}$, algorithm $\mathcal{B}$ works as follows.

Extract: For an extract query on some key y_j, algorithm $\mathcal{B}$ finds the tuple (a_j, r_j, y_j) in the K-List. If $a_j = 1$, $\mathcal{B}$ reports Failure and terminates. Otherwise, it returns r_j as the private key for y_j.

Hash: To respond to hash queries, $\mathcal{B}$ maintains another table called the H-List (having up to $n(q_h + q_s)$ entries). Each entry i in the list is of the form,

$$(m_i, L_i, b_i, u_i, h_i, \gamma_i) \quad \in \quad \Sigma^* \times L \times \{0,1\} \times \mathbb{Z}_q^* \times G_1 \times G_1,$$

and can be interpreted using Table 1.

Entry i	$b_i = 0$	$b_i = 1$
$u_i = 0$	$h_i = \mathcal{H}(m_i, L_i)$ $\gamma_i =$ chain signature on (m_i, L_i)	$h_i = \mathcal{H}(m_i, L_i)$ $\gamma_i =$ chain signature on $(m_i, L_i)/{h_j}^x$ for some $(m_j, L_j, b_j, u_j, h_j, \gamma_j) \in$ H-List such that $m_i = m_j \wedge L_j \prec L_i \wedge b_j = 1 \wedge u_j > 0$
$u_i > 0$	$h_i = g^{u_i}/h_j$ for some $(m_j, L_j, b_j, u_j, h_j, \gamma_j) \in$ H-List such that $m_i = m_j \wedge L_j \prec L_i \wedge b_j = 1 \wedge u_j > 0$ $\gamma_i =$ chain signature on (m_i, L_i)	$h_i = \mathcal{H}(m_i, L_i)$ $\gamma_i =$ chain signature on $(m_i, L_i)/{h_i}^x$
$u_i > 1$		*Sign* query issued to $\mathcal{C}$ on (m_i, L_i)

Table 1: H-List interpretation table.

.

For a hash query j on m_j^*, algorithm $\mathcal{B}$ first parses m_j^* as (m_j, L_j) and scans the H-List for the the unique entry $(m_j, L_j, b_j, u_j, h_j, \gamma_j)$. If such an entry exists, $\mathcal{B}$ returns h_j as its response to the hash query. Otherwise, $\mathcal{B}$ adds the entry $(m_j, L_j, b_j, u_j, h_j, \gamma_j)$ to the H-List as follows.

- First it parses L_j as $\big\langle y_{j(1)}, y_{j(2)}, \ldots y_{j(|L_j|)} \big\rangle$ and scans the K-List to find the entry (a_l, y_l, r_l) such that $y_l = y_{j(|L_j|)}$.
- If $|L_j| > 1$ then $\mathcal{B}$ constructs the sequence $L_j' = \big\langle y_{j(1)}, y_{j(2)}, \ldots y_{j(|L_j|-1)} \big\rangle$ and simulates a *Hash* query to itself on the value (m_j, L_j').

- Let $(m_j, L'_j, b'_j, u'_j, h'_j, \gamma'_j)$ be the entry in the H-List corresponding to (m_j, L'_j) whenever $|L_j| > 1$.

Algorithm $\mathcal{B}$ uses Table 2 to compute its response. It adds $(m_j, h_j, b_j, u_j, h_j, \gamma_j)$ to the H-List and returns h_j as its response to the hash query.[2]

IF	$\lvert L_j \rvert > 1$		$\lvert L_j \rvert = 1$
$a_l = 0$	$h_j \leftarrow \mathcal{H}(m_j, L_j)$ $b_j \leftarrow b'_j; u_j \leftarrow 0$ $\gamma_j \leftarrow \gamma'_j \cdot {h_j}^{r_l}$		$h_j \leftarrow \mathcal{H}(m_j, L_j)$ $b_j \leftarrow 0; u_j \leftarrow 0$ $\gamma_j \leftarrow {h_j}^{r_l}$
$a_l = 1$	$b'_j = 0$	$h_j \leftarrow \mathcal{H}(m_j, L_j)$ $b_j \leftarrow 1; u_j \leftarrow 1$ $\gamma_j = \gamma'_j \cdot {h_j}^{r_l}$	$h_j \leftarrow \mathcal{H}(m_j, L_j)$ $b_j \leftarrow 1; u_j \leftarrow 1$ $\gamma_j \leftarrow {h_j}^{r_l}$
	$b'_j = 1$	$b_j \leftarrow 0$; $u_j \overset{R}{\leftarrow} \mathbb{Z}_q^*$ $h_j \leftarrow g^{u_j}/h_k$, where, $(m_k, L_k, b_k, u_k, h_k, \gamma_k) \in$ H-List such that $m_k = m_j \wedge L_k \prec L_j \wedge b_k = 1 \wedge u_k > 0 \wedge L_k$ is the **largest** sequence $\gamma_j \leftarrow \gamma'_j \cdot y^{u_j} \cdot {h_j}^{r_l}$	

Table 2: H-List computation table.

.

ChainSign: For each chain sign query i $(1 \leq i \leq q_s)$ on $(m_{s(i)}, L_{s(i)})$, algorithm $\mathcal{B}$ scans the H-List to find the unique entry $(m_j, L_j, b_j, u_j, h_j, \gamma_j)$ such that $(m_j, L_j) = (m_{s(i)}, L_{s(i)})$. If such an entry does not exist, $\mathcal{B}$ adds it by simulating a hash query on the message-sequence $(m_{s(i)}, L_{s(i)})$.

1. If $b_j = 1$, algorithm $\mathcal{B}$ finds the entry $(m_k, L_k, b_k, u_k, h_k, \gamma_k) \in$ H-List such that $(m_k = m_{s(i)}) \wedge (L_k \prec L_{s(i)}) \wedge (b_k = 1) \wedge (u_k > 0) \wedge (L_k$ is the **largest** sequence).[3]

- If $u_k = 1$, $\mathcal{B}$ sets $u_k \leftarrow 2$ and updates the H-List tuple $(m_k, L_k, b_k, u_k, h_k, \gamma_k)$.

$\mathcal{B}$ then sets $\sigma_{s(i)} \leftarrow \gamma_j \cdot Sign(m_k, L_k)$ by making a sign query to $\mathcal{C}$ and returns $\sigma_{s(i)}$ as its response to the chain sign query. Note that $Sign(m_k, L_k) = {h_k}^x$

2. If $b_j = 0$, algorithm $\mathcal{B}$ sets $\sigma_{s(i)} \leftarrow \gamma_j$ and returns $\sigma_{s(i)}$ as its response to the chain sign query.

In either case it can be verified that $\sigma_{s(i)}$ is a valid chain signature on $(m_{s(i)}, L_{s(i)})$.

Output. Finally $\mathcal{A}$ outputs a message-chain-signature pair $(m_A, (\sigma_A, L_A))$.

Algorithm $\mathcal{B}$ ensures that an entry for the pair (m_A, L_A) exists in the H-List. (If necessary, by simulating a hash query on (m_A, L_A).)

Result. If $(m_A, (\sigma_A, L_A))$ is not a winning configuration (by adversary $\mathcal{A}$) of Game 1, Algorithm $\mathcal{B}$ reports Failure and terminates.

We know that $(m_A, (\sigma_A, L_A))$ is a winning configuration of Game 1. Algorithm $\mathcal{B}$ finds the entry $(m_A, L_A, b_A, u_A, h_A, \gamma_A)$ in the H-List.

1. If $b_A = 0$, algorithm $\mathcal{B}$ reports Failure and terminates.

[2] **Notes:**

1. $\mathcal{B}$ will simulate the hash function $\mathcal{H}$ by making hash queries to $\mathcal{C}$.
2. If $|L_j| = 0$, $\mathcal{B}$ simply replies with $h_j = \mathcal{H}(m_j)$ without storing the value in the H-List.
3. $\mathcal{B}$'s replies to $\mathcal{A}$'s hash queries are either real or random. Therefore, the hash simulation provided by $\mathcal{B}$ is perfect.

[3] It is possible that $j = k$. By analyzing Table 2, we can conclude that such an entry must necessarily exist.

2. We know that $b_A = 1$. $\mathcal{B}$ finds the entry $(m_k, L_k, b_k, u_k, h_k, \gamma_k) \in$ H-List such that $(m_k = m_A) \wedge (L_k \prec L_A) \wedge (b_k = 1) \wedge (u_k > 0) \wedge$ (L_k is the **largest** sequence).[4]
   If $u_k > 1$, $\mathcal{B}$ reports Failure and terminates.

We know that

$$b_A = 1 \wedge u_k = 1 \tag{1}$$

Therefore, by definition, $\gamma_A = \sigma_A / {h_k}^x$, and $h_k = \mathcal{H}(m_k, L_k)^x$. In other words, σ_A/γ_A is a valid BLS signature under public key y on the message (m_k, L_k).

For any key $y_l \in Y$, define a_l to be the first element of the entry (a_l, r_l, y_l) in the K-List corresponding to public key y_l. Observe that $b_A = 1$ implies that there are an odd number of keys $y_l \in L_A$ such that $a_l = 1$, while $u_k = 1$ implies that there are an even number of keys $y_l \in L_k$ such that $a_l = 1$. Since $u_k = 1$, $\mathcal{B}$ did **not** make a sign query to $\mathcal{C}$ on the message (m_k, L_k).[5]

Algorithm $\mathcal{B}$ returns $(\sigma_A/\gamma_A, (m_k, L_k))$ to $\mathcal{C}$, thereby winning Game 2.

Probability: We need the probability that $\mathcal{B}$ wins Game 2. Consider the following events:

$\mathcal{E}_1$: $\mathcal{B}$ does not abort as a result of $\mathcal{A}$'s extraction queries.
$\mathcal{E}_2$: $\mathcal{A}$ wins Game 1.
$\mathcal{E}_3$: $\mathcal{B}$ does not output Failure during the Result phase.

Clearly, $\epsilon' = \Pr[\mathcal{E}_3 \wedge \mathcal{E}_2 \wedge \mathcal{E}_1] = \Pr[\mathcal{E}_3 | \mathcal{E}_2 \wedge \mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 | \mathcal{E}_1] \cdot \Pr[\mathcal{E}_1]$. The following three claims give the bound on ϵ'.

Claim 1. $\Pr[\mathcal{E}_3 | \mathcal{E}_2 \wedge \mathcal{E}_1] \geq \dfrac{1}{2q_e}$

Proof. Assume that $q_e > 1$. Using Equation 1, define the events,

$\mathcal{E}_4 : b_A = 1$
$\mathcal{E}_5 : u_k = 1$

$\therefore \Pr[\mathcal{E}_3 | \mathcal{E}_2 \wedge \mathcal{E}_1] = \Pr[\mathcal{E}_4 \wedge \mathcal{E}_5 | \mathcal{E}_2 \wedge \mathcal{E}_1] = \Pr[\mathcal{E}_4 | \mathcal{E}_5 \wedge \mathcal{E}_2 \wedge \mathcal{E}_1] \cdot \Pr[\mathcal{E}_5 | \mathcal{E}_2 \wedge \mathcal{E}_1]$.

$$\text{Clearly,} \qquad \mathcal{E}_4 \Rightarrow \sum_{y_l \in L_A} a_l \equiv 1 \pmod 2, \qquad \text{and} \qquad \mathcal{E}_5 \Rightarrow \sum_{y_l \in L_k} a_l \equiv 0 \pmod 2$$

Now consider the Result Section of Game 1 (reproduced below):

(a) i. $L_A \in L$ and the **ChainVerify** algorithm accepts $(m_A, (\sigma_A, L_A))$ as valid.
   ii. No chain sign query has been previously made on the pair (m_A, L_A).
   iii. At least one private key corresponding to L_A has not been extracted.

(b) For **each** chain sign query i, if $(m_{s(i)} = m_A) \wedge (\{L_A, L_{s(i)}\}$ overlap), then there is at least one key in $(L_{s(i)} \cup L_A) \backslash (L_{s(i)} \odot L_A)$ which has not been extracted.

Since, we know that $(m_A, (\sigma_A, L_A))$ is a winning configuration of Game 1, therefore, for all i $(1 \leq i \leq q_s)$ and all $L_{s(i)}$ such that $(m_A = m_{s(i)}) \wedge (L_A \odot L_{s(i)} \neq \emptyset)$, the following two variables are **independent** of $\mathcal{A}$'s view:

$$\sum_{y_l \in L_A} a_l \qquad \text{and} \qquad \sum_{y_l \in L_{s(i)}} a_l$$

[4] It is possible that $L_k = L_A$. By analyzing Table 2, we can conclude that such an entry must necessarily exist.

[5] To visualize this, consider the example given in Section 3.4 with $n = 7$. Assume that $b_5 = b_7 = 1$, while $b_i = 0 \; \forall \; y_i \; (i \neq 5, 7)$. In other words, keys y_5 and y_7 are "fake", while the others are "real". Since $\mathcal{A}$ never obtained a chain signature on (m_A, L'_A) such that $(L'_A \prec L_A) \wedge (L'_A$ contains an odd number of fake keys), therefore, $\mathcal{B}$ never made a sign query to $\mathcal{C}$ on the message $(m_A, \langle y_1, y_2, y_3, y_4 \rangle)$.

which implies that $\sum_{y_l \in L_k} a_l$ is also independent of $\mathcal{A}$'s view.

$\therefore \quad \Pr[\mathcal{E}_4 | \mathcal{E}_5 \wedge \mathcal{E}_2 \wedge \mathcal{E}_1] = \Pr[\mathcal{E}_4 | \mathcal{E}_2 \wedge \mathcal{E}_1]$
$\therefore \quad \Pr[\mathcal{E}_3 | \mathcal{E}_2 \wedge \mathcal{E}_1] = \Pr[\mathcal{E}_4 | \mathcal{E}_2 \wedge \mathcal{E}_1] \cdot \Pr[\mathcal{E}_5 | \mathcal{E}_2 \wedge \mathcal{E}_1]$
It can be shown (using standard techniques) that for all $q_e > 1$,

$$\Pr[\mathcal{E}_4 | \mathcal{E}_2 \wedge \mathcal{E}_1] \geq \frac{1}{q_e}$$

$$\Pr[\mathcal{E}_5 | \mathcal{E}_2 \wedge \mathcal{E}_1] \geq \frac{1}{2}$$

from which we get the required bound. □

Claim 2. $\Pr[\mathcal{E}_2 | \mathcal{E}_1] \geq \epsilon$

Proof. If $\mathcal{B}$ did not abort as a result of $\mathcal{A}$'s extract queries, then the simulation provided by $\mathcal{B}$ is indistinguishable from a real game. Consequently, the probability of $\mathcal{A}$ winning the simulated Game 1 is the same as the probability of $\mathcal{A}$ winning Game 1. Thus, $\Pr[\mathcal{E}_2 | \mathcal{E}_1] \geq \epsilon$. □

Claim 3. $\Pr[\mathcal{E}_1] \geq \frac{1}{e}$

Proof. The probability that $\mathcal{B}$ aborts on the i^{th} extract query on y_i depends on the value a_i of the entry (a_i, r_i, y_i) in the H-List. Consider the event,

$\mathcal{E}_i^e = \mathcal{B}$ does not abort on the i^{th} extract query

$\therefore \quad$ If $j > 1$, then, $\Pr[\mathcal{E}_j^e | \mathcal{E}_{j-1}^e] = \Pr[a_* = 0] \qquad$ (For some a_* in the K-List.)

Since $\mathcal{B}$ did not abort as a result of $\mathcal{A}$'s $j - 1^{th}$extract query, the value a_* must be independent of $\mathcal{A}$'s view until now, and so,

$\Pr[\mathcal{E}_j^e | \mathcal{E}_{j-1}^e] = \Pr[\mathcal{E}_1^e] = \Pr[a_1 = 0] = 1 - \frac{1}{q_e}$

Therefore,

$$\Pr[\mathcal{E}_1] = \left(1 - \frac{1}{q_e}\right)^{q_e} \geq \frac{1}{e},$$

applying the induction hypothesis on q_e extract queries. □

Combining Claims 1, 2 and 3 above, we get the bound on ϵ'.

Hash queries to $\mathcal{C}$: For each entry in the H-List, $\mathcal{B}$ makes at most one hash query to $\mathcal{C}$. Also, $\mathcal{A}$ can make up to q_h hash queries on arbitrary message-sequence pairs (m_*, L_*), and each sequence L_* may contain up to n keys (and therefore, up to n sub-sequences). Consequently, each hash query by $\mathcal{A}$ can cause at most n entries to be added to the H-List. Additionally, adversary $\mathcal{A}$ may make sign queries on q_s distinct message-sequence pairs (m_*, L_*) without making any hash queries on them. These sign queries may cause up to nq_s more entries to be added to the H-List. Thus, for a total of q_h hash queries and q_s sign queries, the number of entries in the H-List (and the number of hash queries made by algorithm $\mathcal{B}$ to challenger $\mathcal{C}$) is upper-bounded by $n(q_h + q_s)$.

Sign queries to $\mathcal{C}$: For each chain chain sign query by adversary $\mathcal{A}$, algorithm $\mathcal{B}$ makes at most one sign query to challenger $\mathcal{C}$. Therefore, $q'_s \leq q_s$.

Running time of $\mathcal{B}$: It only remains to bound the running time t' of $\mathcal{B}$. This is the running time of $\mathcal{A}$, plus the time required to add up to n entries in the K-List and up to $n(q_h + q_s)$ entries in the H-List. Each signature query involves up to one multiplication in G_1. Assuming that the lists are efficiently indexed, the time for searching the H-List and K-List can be ignored. Adding each entry in the H-List and K-List requires 1 exponentiation and up to 1 multiplication in G_1. Therefore, for a maximum of

$n(q_h + q_s)$ entries in the H-List, a maximum of n entries in the K-List, and a maximum of q_s signature queries, we have $t' \leq t + c_{G_1}(n(q_h + 1) + q_s(n + 1))$, where c_{G_1} is the time for 1 exponentiation and 1 multiplication in G_1.

This completes the proof of Theorem 4.4. □

4.4 Adaptive Security In the Chosen Key Model

We note that above construction of chain signatures is also secure in the sense of *adaptive chosen key attacks*, where the adversary includes randomly chosen public keys in the chain signature of the Output phase of Game 1, provided that at least one of the keys in the chain signature is authentic (and not extracted). However, the security definition becomes complicated. To see how the the above construction is also secure in the chosen key model, observe that any chain signature $(\sigma_*, (m_*, L_*))$ is also an **aggregate** signature on some (distinct) messages under public keys $y_* \in L_*$. It is proved in [3, Theorem 3.2], that the aggregate signature scheme is secure under adaptive chosen key and chosen message attacks, provided that all the messages in the signature of the Output phase of Game 1 are distinct. Clearly, for any output chain signature $(\sigma_A, (m_A, L_S))$, it must necessarily hold that all the messages signed under the individual keys $y_l \in L_A$ are distinct.

4.5 Chain Signatures On Distinct Messages

In our model, chain signatures are defined only in a situation when many users sign the same message. However, in many situations users may need to sign different messages and still enjoy the benefits of chain signatures. This is not a major problem and we describe two approaches to solve it. The first and obvious approach is to simply include the individual messages in the hash before the chain signature is computed. However, the security reduction becomes quite complicated in this case. We suggest an alternate and simpler approach that does not require any modification of the chain signature protocol. The idea is to use chain signatures (on a random message) to authenticate the path independently of the actual message(s) in question and then link the message from the chain signature to the actual message(s) using any standard signature scheme that provides non-repudiation. This is the approach we will follow in the example of Section 5.1.1.

4.6 Efficiency

Since for typical security, the value $q = |G_1|$ will be roughly 171 bits, elements of G_1 can be represented in at most 22 bytes. Consequently, the keys and signatures will be at most 22 bytes. The benchmarks of [13] indicate that each pairing operation using these parameters takes $\approx$ 8.6ms and each elliptic curve point exponentiation takes $\approx$ 1.5 ms. These results were obtained on a desktop PC with an AMD Athlon 2100+ 1.8 GHz, 1 GB RAM and an IBM 7200 RPM, 40 GB, Ultra ATA/100 hard drive [13]. Using these values and neglecting the faster operations, we obtain the following performance estimates of the above protocol (assuming n users in the chain):

1. **Signing**: one exponentiation in G_1, one multiplication in G_1, and one computation of $\mathcal{H}$ (total $<$ 2ms).
2. **Verification**: n pairing computations and multiplications in G_2, and n computations of $\mathcal{H}$ (giving $<$ 1 second for $n = 100$).

5 Applications of Chain Signatures

Considering that chain signature enable us to correctly validate the path of any received message using very short signatures and provides non-repudiation, we can consider several applications: mobile agent authentication [14, 15], group e-commerce (e-commerce transactions where multiple entities are involved such that direct interaction is not possible between many of them), electronic work-flow enforcement (ensuring the order in which participants should be involved), secure routing, authenticated mail relaying, IP tracing, Grid computing and Mobile IP. Here, we will discuss one such application.

5.1 Stateless Routing

The most common and robust interior and exterior routing protocol is the Border Gateway Protocol (BGP) [16, 17], which is essentially a **Path Vector Routing** protocol.[6] In this protocol, routers repeatedly advertise 'better' routes (along with the path details) to their immediate neighbors. On receiving an update, a router checks its routing table to decide if this advertised route is better than its existing routes. If so, the router updates its table and advertises the new updated route (from *its own tables*) to all its immediate neighbors except the one from which this update was received. Also, an update is discarded once it reaches a preset maximum 'cost'. Although BGP is very robust, it has many security vulnerabilities [18]. For instance, an adversary could send a forged route advertisement and corrupt legitimate routing tables. Such security issues can lead to drastic consequences [19]. The important thing to note here is that a routing table is updated every time a better route advertisement is received. Even if the actual update is not forged, a rogue router may extract intermediate routes from real updates and claim to have a shorter route, which we call a *Path Extraction Attack*. A key security issue here, therefore, is to ensure the authenticity of the (path of the) received advertisement before accepting it as valid. This is best explained using an example.

Consider the network of routers given in Figure 2. The numbers on the links indicate the metric. Assume that E is a rogue router who would like to intercept traffic sent from D to A, which would ordinarily be routed directly via router C.

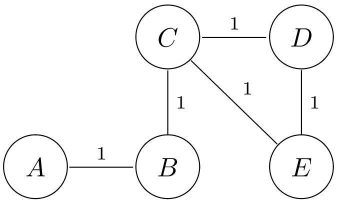


Figure 2: Scenario for a Path Extraction Attack

In BGP, assume that (ordinary) signatures are used to authenticate updates. Then the following update messages will be sent. (Consider only routes to A. The symbol $\triangleleft$ denotes a path)

Example 1. BGP updates

1. $A \rightarrow B$: $\{Sign_A(A\triangleleft)\}$
2. $B \rightarrow C$: $\{Sign_A(A\triangleleft),\ Sign_B(A \triangleleft B)\}$
3. $C \rightarrow D, E$: $\{Sign_A(A\triangleleft),\ Sign_B(A \triangleleft B),\ Sign_C(B \triangleleft C)\}$
4. $D \rightarrow E$: $\{Sign_A(A\triangleleft),\ Sign_B(A \triangleleft B),\ Sign_C(B \triangleleft C),\ Sign_D(C \triangleleft D)\}$
5. $E \rightarrow D$: $\{Sign_A(A\triangleleft),\ Sign_B(A \triangleleft B),\ Sign_C(B \triangleleft C),\ Sign_E(C \triangleleft E)\}$

In this setup, traffic originating from D and destined for A will never pass through E. However, if E wants, it can simply extract the last two signatures in the update received from C and claim (to D) that it has a direct route to A by sending it the update $\{Sign_A(A),\ Sign_E(A \triangleleft E)\}$. In this case, D will assume that E has a more efficient path to A and use it to forward that traffic. Secure BGP (S-BGP) [20] mitigates this attack by requiring that "links" in the updates be two-way (i.e. contain signatures from each endpoint authenticating the other). In S-BGP, the routing updates would be as follows:

Example 2. S-BGP updates

[6] That is, a routing protocol which maintains the path that update information takes as it diffuses through the network.

1. $A \rightarrow B:\quad \{Sign_A(A \triangleleft B)\}$
2. $B \rightarrow C:\quad \{Sign_A(A \triangleleft B),\ Sign_B(B \triangleleft C)\}$
3. $C \rightarrow D:\quad \{Sign_A(A \triangleleft B),\ Sign_B(B \triangleleft C),\ Sign_C(C \triangleleft D)\}$

   $C \rightarrow E:\quad \{Sign_A(A \triangleleft B),\ Sign_B(B \triangleleft C),\ Sign_C(C \triangleleft E)\}$
4. $D \rightarrow E:\quad \{Sign_A(A \triangleleft B),\ Sign_B(B \triangleleft C),\ Sign_C(C \triangleleft D),\ Sign_D(D \triangleleft E)\}$
5. $E \rightarrow D:\quad \{Sign_A(A \triangleleft B),\ Sign_B(B \triangleleft C),\ Sign_C(C \triangleleft E),\ Sign_E(E \triangleleft D)\}$

Although the protocol of Example 2 is secure from the path extraction attack, it has two drawbacks: (1) Each router must be "aware" of its neighbors, and (2) In the example, router C can no longer broadcast the same message to its neighbors. This can cause scalability problems as follows. Firstly, each router must establish authenticity of each of its peer(s). Secondly, each update is peer-specific and therefore, even a single path change could result in a large number of messages sent by a host with many neighbors. It would be much simpler if the underlying routing protocol resisted path extraction attacks and required each router to broadcast only one short message on each update without being aware of its neighbors (as in Example 1). We call such a protocol a **Stateless Routing Protocol**. Such stateless-ness is useful if path vector routing is used over broadcast networks (such as ad-hoc wireless/sensor networks).

Current research on S-BGP authentication assumes the above stateful scenario of Example 2 and is focused on methods to reduce the number of signatures transmitted and/or processing time for signing and verification. For instance, aggregate signatures have been proposed to keep the signature payload to a constant size [3]. The authors of [20] propose the use of *Signature Amortization* [21] coupled with aggregate or sequential aggregate signatures [4] to reduce the size of update messages and the number of signatures. However, all the above works assume some sort of stateful environment, where information about peer is pre-distributed or known. In this work, we focus on how to achieve security under path extraction attacks in a stateless implementation of BGP. Our proposed protocol, called Stateless Secure-BGP (SS-BGP) is based on chain signatures and provides the following benefits.

1. It is fully stateless. Routers need not be aware of their neighbors.
2. The update size is constant irrespective of the number of peers. Additionally, only one message needs to be transmitted if using broadcast.
3. The signing time is constant. The verification time is linear to the size of the path and is comparable to efficient stateful S-BGP based on aggregate signatures [20].

5.1.1 Stateless S-BGP (SS-BGP)

In this implementation, we will assume the same routing logic of BGP. However, we will use Example 1 in our scenarios and assume that routers may not be aware of their immediate neighbors. We will assume that routers can be directly identified using their public keys.

1. Let Sign_i, Verify_i denote sign and verify functions of user i under another existentially unforgeable signature scheme, such as BLS.
2. Denote by y_i the public key of user i under a chain signature scheme.
3. Denote by $L_i = \langle y_1, y_2, \ldots y_n \rangle$ some ordered sequence of (public keys of) routers that would be affected by a given routing update. Note that there will be many such distinct sequences for the same update and y_1 would be the first name (i.e., key) all these sequences.
4. Each individual router may want to add additional information to the update. Denote this additional information by router i (intended for routers $i + 1$) by M_i.
5. Denote by U_i the update message of i that is sent to $i + 1$ describing this update.

The SS-BGP protocol is as follows.

Initialize First, the initiator, 1 generates a message $M \in \{0,1\}^*$ describing this update (i.e., the name of the Originating AS and a time-stamp). Let M_1 be the additional information (if any). To start the update user 1 computes $Sig_1 = \mathsf{Sign}_1(M, M_1)$ and $(\sigma_1, (M, L_1))$, a chain signature on (M, L_1). It broadcasts the update $U_1 = (\sigma_1, M, L_1, M_1, Sig_1)$ to all its neighbors.

Update On receiving update $U_i = (\sigma_i, M, L_i, M_i, Sig_i)$, router $i+1$ does the following.

Accept Router i accepts this update if the following checks pass (and aborts otherwise):

1. $(\sigma_i, (M, L_i))$ is a valid chain signature.
2. $\mathsf{Verify}_i(Sig_i, (M, M_i)) = \mathsf{True}$.
3. The destination and time-stamp defined in M are correct.
4. Routes to each of the links specified in L_i exist and all the nodes in L_i are trusted.

It then checks M_i for additional information regarding this update (if any).

Propagate If the update is valid, router $i+1$ propagates it as follows:

1. It constructs the sequence L_{i+1} by appending its own public key y_i to L_i and computes a chain signature $(\sigma_{i+1}, (M, L_{i+1}))$ using σ_i and its private key x_{i+1}.
2. It constructs a message M_{i+1} with additional routing information (if any) and computes the signature $Sig_{i+1} = \mathsf{Sign}_{i+1}(M, M_{i+1})$.
3. It broadcasts the update $U_{i+1} = (\sigma_{i+1}, M, L_{i+1}, , M_{i+1}, Sig_{i+1})$.

Let us analyze this method:

1. *Security*: The use of chain signatures ensures that router $i+1$ cannot prove a direct route to any router $j < i$ without access to the chain signature sent by j. Consequently, path extraction attacks are infeasible. The use of the time-stamp avoids any replay attacks. The use of Sig_i ensures that some correlation is maintained between an advertised route and the actual destination.

2. *Storage*: To be able to validate the signatures, each host must be able to store/obtain public keys of all routers in question, which may lead to scalability problems. This problem is easily solved using Identity Based Chain Signatures (IBCS) (briefly discussed in the conclusion) and Identity Based Signatures (IBS) where the IP address of a host acts as the public key. The security model of IBCS would be identical to the model described here.

3. *Overhead*: The overhead incurred by (M, M_i, Sig_i) in update U_i cannot be avoided. The chain signature additionally incurs the overhead of (σ_i, L_i). Assuming that public keys can be uniquely identified by IP addresses, the sequences L_i can be constructed from the IP addresses of the nodes in the path. Consequently, L_i is part of the update message itself and does not incur any overhead. The only overhead is then the size of a chain signature, which will be less than 22 bytes using the parameters of [3].

4. *Multiple Updates Aggregation*: In the above description, we assumed that each advertisement U_i contains only one route and is transmitted instantaneously. In the real world, each advertisement contains multiple routes and is sent periodically. Fortunately, both the chain signature and individual signature schemes used above allow for *signature aggregation* and *aggregate verification* where a large number of (chained or individual) signatures can be verified at once [3].[7]

[7] The use of aggregate signatures in BGP advertisement verification has already been discussed in [3]. We additionally suggest using chain signatures for increased scalability.

6 Summary

In this paper, we introduced the notion of Chain Signatures as an extension of Boneh et al.'s short signatures [9]. Although chain signature arise naturally from the aggregate signatures of [3] due to the inherent properties of bilinear maps, the security requirements of chain signatures is significantly different as demonstrated in Sections 3.2 and 3.4. We note that chain signatures without using bilinear maps were independently proposed in [15, 22] in which the authors used hypothetical primitives called *Strong Associative One-Way Functions* (SAOWFs).

The protocol presented here uses a standard certificate-based PKI. However, it is possible to construct Identity Based Chained Signatures (IBCS) because of the observation that the Identity Based Signature (IBS) schemes of [23, 24] support signature aggregation with the property that once aggregated, individual signatures cannot be extracted.

Considering that chained signatures enable us to correctly validate the path of any received message and provide non-repudiation, we can consider several applications: mobile agent authentication [22, 15], electronic auctions, relaying, token based authentication. As a practical demonstration of applications, we presented a novel method for stateless routing.

The main feature of chain signatures that distinguishes them from other multi-user signature schemes is that chain signatures provide *delete-protection* (See Section 2.1). The chain signature scheme presented here, however, does not provide *strong delete-protection.* Signatures that also provide strong delete protection are called *Strong Chain Signatures* (SCS). However, whether practical SCS schemes exist or not, is still an open question at this stage.

References

[1] Masahiro Mambo, Keisuke Usuda, and Eiji Okamoto. Proxy signatures for delegating signing operation. In *CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security*, pages 48–57, New York, NY, USA, 1996. ACM Press.

[2] Silvio Micali and Ronald L. Rivest. Transitive signature schemes. In *CT-RSA*, pages 236–243, 2002.

[3] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.

[4] H. Shacham. Sequential aggregate signatures from trapdoor homomorphic permutations, 2003.

[5] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. Cryptology ePrint Archive, Report 2006/096, 2006.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[7] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In *PKC '03: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46, London, UK, 2003. Springer-Verlag.

[8] Mike Burmester, Yvo Desmedt, Hiroshi Doi, Masahiro Mambo, Eiji Okamoto, Mitsuru Tada, and Yuko Yoshifuji. A structured elgamal-type multisignature scheme. In *PKC '00: Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography*, pages 466–483, London, UK, 2000. Springer-Verlag.

[9] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532, London, UK, 2001. Springer-Verlag.

[10] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[11] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 354–368, London, UK, 2002. Springer-Verlag.

[12] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996.

[13] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. Cryptology ePrint Archive, Report 2005/028, 2005.

[14] Amitabh Saxena and Ben Soh. A mobile agent authentication protocol using signature chaining with bilinear pairings. Cryptology ePrint Archive, Report 2005/272, 2005.

[15] Amitabh Saxena and Ben Soh. Authenticating mobile agent platforms using signature chaining without trusted third parties. In *Proceedings of The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05)*, pages 282–285, Hong kong, 2005. IEEE computer press.

[16] Y. Rekhter, T. Li, and S.Hares. RFC 4271: A Border Gateway Protocol 4 (BGP-4), January 2006.

[17] Jennifer Rexford, Jia Wang, Zhen Xiao, and Yin Zhang. Bgp routing stability of popular destinations. In *ACM SIGCOMM IMW (Internet Measurement Workshop) 2002*, 2002.

[18] S. Murphy. RFC 4272: BGP Security Vulnerabilities Analysis, January 2006.

[19] R. Mahajan, D. Wetherall, and T. Anderson. Understanding bgp misconfiguration, 2002.

[20] Meiyuan Zhao, Sean W. Smith, and David M. Nicol. Aggregated path authentication for efficient bgp security. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 128–138, New York, NY, USA, 2005. ACM Press.

[21] Jung Min Park, Edwin K. P. Chong, and Howard Jay Siegel. Efficient multicast packet authentication using signature amortization. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 227, Washington, DC, USA, 2002. IEEE Computer Society.

[22] Amitabh Saxena and Ben Soh. A novel method for authenticating mobile agents with one-way signature chaining. In *Proceedings of The 7th International Symposium on Autonomous Decentralized Systems (ISADS 05)*, pages 187–193, China, 2005. IEEE Computer Press.

[23] Jae Choon Cha and Jung Hee Cheon. An identity-based signature from gap Diffie-Hellman groups. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 18–30. Springer, 2003.

[24] B. Libert and J. Quisquater. The exact security of an identity based signature and its applications. Technical Report 2004/102, Cryptology ePrint Archive, 2004.