

# One-Way Signature Chaining - A New Paradigm For Group Cryptosystems

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

February 23, 2007

## 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*. In this paper, we 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.

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, 4] and sequential aggregate signatures [5, 6].

The rest of this paper is organized as follows. We give the background and motivation for chain signatures in Sections 2 and 3. We then formalize the intuition of chain signatures in Section 4.2. Finally, in Section 5 we present the scheme and prove its security in Section 5.3. We then discuss some applications of chain signatures in Section 6.

## 2 Background

STRUCTURED SIGNATURES. When multiple entities sign a document, the signing order often reflects the role of each signer and signatures with different signing orders are regarded as multisignatures with

different meanings. Although the signing order is of little relevance to authentication, there are other aspects one should take into account, such as the liability of the signers which may be determined by the signing order. Moreover, each signer may only wish to sign after the previous signers have done so or only if certain users have **not** signed. Additionally, the verifier may require that the correct order has been adhered to. A multisignature scheme is said to be *structured* if the group of signers is structured [7]. The structure takes into account the signing order of the entities. Strictly speaking, there are two types of structured signatures: (a) *Serial*, where the order of every member is specified. (b) *Parallel*, where the order of some members is hidden [7]. In this work, by structured signatures, we always imply serial structured signatures. However, our construction of structured signatures (called *chain signatures*) can be converted into a parallel one simply by hiding the order of some signers (i.e., by not including their public keys in the sequences before computing the chain signatures).

**REQUIREMENTS OF STRUCTURED SIGNATURES.** The simplest model of structured signatures is when many users to commit to a single contract (we will later extend our model to multiple contracts). By the *view*, of a signer, we imply the information known to the signer at the time of making this contribution. In our case, the view of a signer includes the message to be signed and the names (with the order) of the entities who have already signed the message. In structured signatures, not only should it be infeasible to forge a signature under any user's public key, but also, it should be infeasible to modify the view of any user indicated by the multisignature. We define the following requirement for structured signatures. The definition extends the requirements of (serial) structured signatures of [7] by including truncation resilience, extensibility and non-interactivity.

1. **Unforgeability.** It should be infeasible to contribute to a multisignature under some public key without knowing the corresponding private key.
2. **Extensibility.** It should be possible to extend the signature. That is, new members may contribute at any time.
3. **Non-Interactivity.** It should be possible to extend the multisignature without interaction with any of the previous signers.
4. **Constant Size.** The size of the multisignature at any stage must be constant.
5. **Reorder protection.** It should be infeasible to change the order of the public keys needed for verification of the multisignature from the order indicated by the view of each individual signer.
6. **Truncation Resilience.** It should be infeasible to truncate the view of any signer who has previously contributed to the multisignature. A truncated view is a subset of the real view. Note that truncation is different from reordering.

Any signature scheme is expected to satisfy unforgeability and we will not elaborate on it. Structured signatures satisfying reorder protection have been extensively studied in the literature [7, 8, 9]. Many multisignature schemes (such as [10, 11, 6, 5]) can be modified to achieve extensibility, non-interactivity, constant size and reorder protection. However, there is very little work on signature schemes satisfying truncation resilience [12]. The authors of [12] provide a mobile agent authentication protocol that provides truncation resilience by requiring nodes to sign the names of the next (intended) signer. However, their model of truncation resilience is inherently different from ours because it makes signers liable for an individual who has not yet committed to the contract. Specifically, the view of the signer is different in their model.

We define a *chain signature* to be a structured signature scheme that satisfies all the above properties. To give a clear intuition of chain signatures, we elaborate on the chaining scheme proposed by Karjoth, Asokan and Gülcü for mobile agent authentication [12], which we call the KAG scheme, and explain how our model supersedes theirs.

## 2.1 KAG Authentication Scheme For Mobile Agents

MOBILE AGENTS. Mobile agents are software programs that live in computer networks, performing their computations and moving from host to host as necessary to fulfill their goals [12]. Electronic commerce (such as comparison shopping) appears to be a particularly interesting application for agents. The principal scheme for comparison shopping, described in [12] is the following:

1. The agent owner sets up the agent with a description of the good.
2. The agent travels from the owner's site to the directory in the electronic marketplace, where it obtains the locations of all retailers that offer the good.
3. The agent visits the electronic store-front of each retailer in turn. The agent gives the description of the good to the store-front and is quoted a price.
4. The agent eventually returns to its owner and delivers a report of its findings.

AUTHENTICATION, INTEGRITY AND NON-REPUDIATION. As the agent hops from shop to shop, collecting offers, it builds a chain of offers corresponding to the nodes visited in sequence. In the description,  $O_1, O_2, \dots, O_m$  are the encapsulated offers of the  $m$  shops visited sequentially by the agent. Some shops, but not shop  $S_m$ , may conspire with the attacker, possibly being shop  $S_{m+1}$ . Let  $1 \leq i \leq m$ . The originator of the agent is  $S_0$  and may have offer  $O_0$ . The requirements for signatures given in [12] are:

1. *Non-repudiability*: Shop  $S_i$  cannot repudiate its offer  $O_i$  once it has committed.
2. *Integrity*: None of the encapsulated offers  $O_k$ , where  $k < m$ , can be modified.
3. *Publicly Verifiable Forward Integrity*: Anyone can verify if the chain is valid at  $O_i$ .
4. *Insertion resilience*: An offer can be only be inserted at  $i$  if  $S_i$  is the attacker.
5. *Truncation Resilience*: The chain can only be truncated at  $i$  if  $S_i$  is the attacker.

As noted in [12], the first four properties can be achieved using any standard two-party signature that is existentially unforgeable. The most difficult problem, however, is providing truncation resilience. Let us first discuss why trivial approaches do not work. Denote by  $Sig_i$  the signing function of  $S_i$ . Assume that the agent is carrying signatures  $\{Sig_0(O_0), Sig_1(O_1), \dots, Sig_{m-1}(O_{m-1})\}$  of shops  $\{1, 2, \dots, m-1\}$  on their offers at step  $m$ . To ensure publicly verifiable forward integrity, these signatures must be in cleartext. Consequently,  $m$  can truncate any signature.

THE KAG SCHEME. The following protocol presented in [12] provides a weak form of truncation resilience. In their protocol, each user  $i$  includes  $Sig_i(O_{i-1}, O_i, S_{i+1})$  instead of just  $Sig_i(O_i)$  as in the previous example. Therefore, at step  $m$ , the agent will carry the following signatures

$$\{Sig_0(O_0, S_1), Sig_1(O_0, O_1, S_2), Sig_2(O_1, O_2, S_3), \dots, Sig_{m-1}(O_{m-2}, O_{m-1}, S_m)\}$$

WEAKNESSES IN THE KAG SCHEME. The above protocol has several weaknesses, some noted by the authors themselves [12]. We briefly elaborate on the major ones: (a) The above scheme is reactive because it does not prevent an attacker from truncating the chain - rather, it allows truncation but enables detection. (b) If the attacker is able to corrupt any shop  $S_j$  for  $j < m$ , it can truncate and insert offers after  $j$  even without detection.<sup>1</sup> (c) The identity of  $S_{i+1}$  must be known to  $S_i$  at the time  $S_i$  computes its signature. Consequently,  $S_{i+1}$  cannot remain anonymous from  $S_i$ . (d) If the agent branches at  $i$ , then shop  $S_i$  would have to create a signature as many times as there are branches. (e) The size of the signature payload is linear to the number of nodes.

As noted earlier, the last drawback (regarding the size) can be overcome using aggregate signatures [3] or sequential aggregate signatures [6, 5]. However, none of these signature schemes take into account truncation resilience.

DESIRED PROPERTIES. Assume that a multisignature scheme is used for authenticating mobile agents (unlike the above protocol). In this model, we desire the following properties.

---

<sup>1</sup>By corruption, we imply that the attacker succeeds only in obtaining the private key of  $j$ . However, all messages sent/received by  $j$  when it was uncorrupted are lost.

1. Non-repudiability, Integrity, Publicly Verifiable Forward Integrity as defined earlier.
2. *Truncation and Insertion Resilience*: Let the agent be as it step  $m$ . To truncate or insert at  $i$  (for some  $i < m$ ), the attacker must corrupt *all* shops from  $S_{i+1}$  to  $S_m$ .
3. *Constant Size*: The size of signature payload is constant at any stage  $i$ .
4. *Non-Interactivity*: Any shop  $S_i$  need not know the identity of shop  $S_{i+1}$ .

Property 2 above gives the intuition behind our security model for truncation resilience. Informally, it should be infeasible to truncate the chain at  $i$  (for some  $i < m$ ) even if the private key of  $i$  is known. Additionally, our model of truncation resilience is proactive - the attacker cannot obtain *anything* linking to a truncated view from a given multisignature.

### 3 Intuition Behind Chain Signatures

We give an informal intuition behind the security of chain signatures using the following scenario. Alice signs some message  $m$  and sends the signature  $\sigma_A$  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 cannot deny (or hide) Bob's knowledge of the fact.

More formally, assume that users  $A, B$  compute signatures  $\sigma_A, \sigma_B$  (on the same message) using private keys  $SK_A, SK_B$  respectively. Define the following properties:

1. **Aggregation**: Given signatures  $\sigma_A, \sigma_B$  it is easy to compute a combined signature  $\sigma_{\{A,B\}}$  that can be verified using public keys  $PK_A, PK_B$ .
2. **Truncation Resilience**: Given  $\{\sigma_{\{A,B\}}, PK_A, PK_B\}$ , it must be infeasible to compute  $\sigma_A$  or  $\sigma_B$ .
3. **Strong Truncation Resilience**: This is a stronger variant of the previous property.
  - Given  $\{\sigma_{\{A,B\}}, PK_A, PK_B, SK_A\}$ , it must be infeasible to compute  $\sigma_B$ .
  - Given  $\{\sigma_{\{A,B\}}, PK_A, PK_B, SK_B\}$ , it must be infeasible to compute  $\sigma_A$ .

The above idea can be extended to an arbitrary number of users. Assume that users  $1, 2, \dots, n$  compute signatures  $\sigma_1, \sigma_2, \dots, \sigma_n$  (on the same message) using private keys  $SK_1, SK_2, \dots, SK_n$  respectively. We can then similarly define:

1. **Aggregation**: Given signatures  $\sigma_1, \sigma_2, \dots, \sigma_n$ , it is easy to compute a combined signature  $\sigma_{\{1,2,\dots,n\}}$  that can be verified using public keys  $PK_1, PK_2, \dots, PK_n$ .
2. **Truncation Resilience**: Given  $\{\sigma_{\{1,2,\dots,n\}}, PK_1, PK_2, \dots, PK_n\}$ , it must be infeasible to compute  $\sigma_\alpha$  for any  $\alpha \subsetneq \{1, 2, \dots, n\}$ .
3. **Strong Truncation Resilience**: Given  $\{\sigma_{\{1,2,\dots,n\}}, PK_1, PK_2, \dots, PK_n, SK_{\beta_1}, SK_{\beta_2}, \dots, SK_{\beta_i}\}$  for  $i < n$  and  $\{\beta_1, \beta_2, \dots, \beta_i\} \subsetneq \{1, 2, \dots, n\}$ , computing any signature  $\sigma_\alpha$  for  $\alpha \subseteq \{1, 2, \dots, n\} \setminus \{\beta_1, \beta_2, \dots, \beta_i\}$  must be infeasible.

A signature scheme that satisfies the aggregation and truncation resilience conditions is called a *Chain Signature* (CS) scheme. A CS scheme that additionally satisfies the strong truncation resilience condition is called a *Strong Chain Signature* (SCS) scheme. 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 truncation resilience. However, it will provide truncation resilience.

## 4 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,  $\langle y_1, y_2, y_3 \rangle$ . The symbol  $\theta$  denotes the empty sequence with zero elements. The symbol  $\diamond$  denotes the empty string of zero length.
2. Let  $L_a = \langle y_1, y_2, \dots, y_k \rangle$  be some non-empty sequence. For any other sequence  $L_b$ , we say that  $L_b \prec L_a$  if and only if  $L_b = \langle y_1, y_2, \dots, y_i \rangle$  and  $0 \leq i \leq k$ . 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.
3. 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  $L_a \odot L_b \neq \emptyset$ .

### 4.1 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  $\tau$  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  $(x_i, y_i) \stackrel{R}{\leftarrow} \text{KeyGen}$  on the  $i^{\text{th}}$  run of this algorithm.

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

1. If  $L_i = \theta$  and  $\sigma_i = \diamond$  the algorithm outputs **VALID** and terminates.
2. If  $L_i = \theta$  and  $\sigma_i \neq \diamond$  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  $(\sigma_j, L_j)$  is a purported chain signature on message  $m$ , and  $L_j = \langle y_1, y_2, \dots, y_j \rangle$  is some sequence of  $j$  public keys such that  $y_i \notin \{y_1, y_2, \dots, y_j\}$ . 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  $(m, \sigma_j, L_j)$  as input (i.e. it checks whether  $(\sigma_j, L_j)$  is a valid chain signature on  $m$  or not). If  $(\sigma_j, L_j)$  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  $(\sigma_j, L_j)$  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  $(\sigma_i, L_i)$  on message  $m$  such that  $L_i = \langle y_1, y_2, \dots, y_j, y_i \rangle$ . It outputs  $(\sigma_i, L_i)$  and terminates.

The **ChainVerify** and **ChainSign** algorithms must satisfy the standard consistency constraint of signatures. That is, if the input  $(m, \sigma_i, L_i)$  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  $\sigma_j = \diamond$  and  $L_j = \theta$ .

## 4.2 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

**Setup.** The challenger  $\mathcal{C}$  sets a parameter  $\tau$  and gives it to the adversary  $\mathcal{A}$ , who then selects a game parameter  $n$ .

On receiving  $n$ ,  $\mathcal{C}$  generates  $n$  key-pairs  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \xleftarrow{R} \mathbf{KeyGen}$  and gives the set  $Y = \{y_1, y_2, \dots, y_n\}$  of  $n$  public keys to  $\mathcal{A}$ . Denote by  $L$  the set of all non-empty sequences with elements from  $Y$ .

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

1. *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.
2. *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  $x_{e(j)}$ .

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

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

- (a) (a)  $L_A \in L$  and the **ChainVerify** algorithm accepts  $(\sigma_A, (m_A, L_A))$  as valid.
  - (b) No chain sign query has been previously made on the pair  $(m_A, L_A)$ .
  - (c) At least one private key corresponding to  $L_A$  has not been extracted.
- (b) For **each** chain sign query  $i$ , if  $(m_{s(i)} = m_A) \wedge (\{L_A, L_{s(i)}\} \text{ overlap})$ , then there is at least one key in  $(L_{s(i)} \cup L_A) \setminus (L_{s(i)} \odot L_A)$  which has not been extracted.

We use the random oracle model [13], 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 4.1.** 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  $\tau$  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  $\epsilon$ . 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.

**Remark:** The reader might ask the following question: “Why do we not allow the adversary to submit a chain signature  $\sigma_i$  on some message-sequence pair  $(m_i, L_i)$  and a public key  $y_{i+1} \notin L_i$ , and require the **ChainSign** oracle to return a chain signature  $\sigma_{i+1}$  on  $(m_i, \langle L_i, y_{i+1} \rangle)$ ? It appears that the adversary of Game 1 is given less powers than a real-world adversary.” However, this is not the case when we observe that the adversary is also allowed to **extract** all the private keys of  $L_i$  and compute the above chain signature  $\sigma_i$  without the help of the **ChainSign** oracle, or make a query to the **ChainSign** oracle on  $(m_i, L_i)$  to obtain  $\sigma_i$ . In other words, the two models are equivalent. The reason for not allowing the above query is to restrict the information (i.e., chain signatures) the adversary has when all the private keys in  $L_i$  are *not* extracted and the adver.

### 4.3 Weak Adaptive Security

Although full adaptive security of chain signatures under known key attacks and chosen message attacks is given by Definition 4.1, we will prove the security of our construction in a restricted model, which we called **weak** adaptive known key and chosen message attacks. In this model, the attacker  $\mathcal{A}$  submits the extract queries before the challenge public keys are generated (for instance by specifying their index numbers). This is a reasonable assumption considering that even though the adversary is not allowed to actively corrupt participants, it is given unlimited access prior to the execution of the session. We define this using a modified version of Game 1, which we call Game 1-a.

**Game 1-a:** This is a variation of Game 1 with the following differences: (1) In the setup phase, the adversary submits not only the parameter  $n$  but also an  $n$  bit string  $extr$ , where the 1’s of  $extr$  denote the indexes of the keys that the adversary wants to extract. (2) In the challenge phase, the challenger responds, not only with the public keys  $Y$ , but also the extracted private keys  $X$  corresponding to the non-zero bits of  $extr$ . (3) There are no private key extract queries in Game 1-a.

**Definition 4.2.** We say that the chain signature scheme is  $(n, \tau, t, q_s, q_e, q_h, \epsilon)$ -secure under a **weak adaptive known-key and chosen-message attack** if, for some parameters  $\tau$  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-a with probability at least  $\epsilon$ . 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. Game 1-a is similar to Game 1, except that the adversary must submit the (indexes of the) private key extract queries before the setup phase is completed.

**Reason For Weakening Game 1:** At first, it may seem strange that we have further weakened Game 1 (which is already weak compared to a game for *chosen key attacks*) to Game 1a. However, we have been unable to prove the security of our presented scheme under Game 1, which is an open problem at this stage. The reader is encouraged to try and prove it, since that would be a non-trivial result.

### 4.4 Analysis Of Game 1

Let us analyze the Result Section of Game 1 (and Game 1a). 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 in Figure 1 of the **ChainSign** algorithm at some stage  $j$ . Since we know that **ChainSign** can be “reversed” from  $\sigma_j$  (using the private key  $x_j$ ) to obtain  $\sigma_{j-1}$ , we can also consider Figure 2 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  $\sigma_A$  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  $\sigma_A$  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.

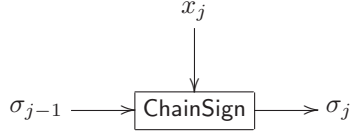


Figure 1: Typical Usage Of ChainSign.

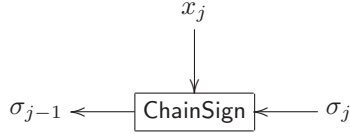


Figure 2: Valid Usage Of ChainSign.

## 4.5 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 [5, 6], multisignatures [11], aggregate signatures [3], and structured multisignatures [7]. We distinguish between two types of forgers for chain signatures.

**Type 1 Forger** This adversary either does not make any non-trivial queries, or for each non-trivial query  $j$  made, 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 chain-signature-message tuple  $(\sigma_A, (m_A, L_A))$  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):

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

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

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

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

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 ([5, 11, 3, 7]) when we keep  $y_4$  as the challenge public key.

## 4.6 Relationship With Verifiably Encrypted Signatures

Although CS are quite different from other signature schemes, they are very similar to *verifiably encrypted signatures* (VES) of [3]. VES can be abstractly described as follows. Suppose Alice wants to send a signature (on a payment order) to Bob in return for some goods bought online. However, she wants to ensure that Bob obtains her signature only after he has fulfilled his commitments. On the other hand, Bob suspects that Alice may deny him the signature after receiving the goods. The problem can be



solved using VES and a trusted third party (the *Adjudicator*). An essential property of the adjudicator’s public key is that existential forgery is easy but standard forgery is hard. Using this scheme, Alice creates her signature  $\sigma_A(m)$  on the message  $m$  (the payment order) under her public key  $A$  and creates an existential forgery under the adjudicator’s VES-public key  $J$  to obtain a forged signature-message pair  $(\sigma_J(m_*), m_*)$ . She combines the two signatures  $\sigma_A(m)$  and  $\sigma_J(m_*)$  to obtain an aggregate signature  $\sigma_{(A,J)}(m, m_*)$  and sends  $(\sigma_{(A,J)}(m, m_*), m, m_*)$  to Bob. VES have the following properties: (1) Bob can verify that  $\sigma_{(A,J)}(m, m_*)$  is an aggregation of the signatures  $\sigma_A(m)$  on  $m$  and  $\sigma_J(m_*)$  on  $m_*$ . (2) Bob cannot extract either signature without the Adjudicator’s (or Alice’s) help. Both these properties when combined are termed *opacity* [3]. From this we can observe the following similarities and differences between VES and CS.

**Similarities.** (1) Both CS and VES are based on the idea of aggregating many signatures into one object. (2) Similar to CS, VES provide truncation resilience in the form of opacity.

**Differences.** (1) VES are defined only for one signer (and possibly many adjudicators [3]) and do not provide any sort of security guarantee when multiple signers are involved. (2) Existential forgery is not possible under any user’s key in CS.

Therefore, chain signatures can be considered as an extension of VES to multiple users. Also, using the idea of an existentially forgeable adjudicator public key, we can think of *verifiably encrypted chain signatures* (*VECS*).

**Remark:** To see why the above definition of CS is non-trivial, observe that the opacity of VES (or unforgeability of aggregate signatures) **does not** imply truncation resilience (the reader is encouraged to try to prove the security of Game 1a under the standard aggregate signature scheme of [3]).

## 5 Chain Signatures Using Bilinear Maps

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 [14, 15, 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.

In a practical implementation,  $G_1$  is a subgroup of the (additive<sup>2</sup>) 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 [14, 15] or the Tate pairing [16]. Typically  $q \geq 2^{171}$  so that the fastest algorithm for computing discrete logarithms in  $G_1$  (Pollard’s rho method [17, p.128]) takes  $\geq 2^{85}$  iterations [14].

### 5.1 BDH Parameter Generator

Using the idea of [14], 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  $\text{CDH}_{(g, G_1)}$ :** Let  $g \stackrel{R}{\leftarrow} G_1$  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.

---

<sup>2</sup>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.

**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  $\mathbf{Exp}_{\mathcal{A}}^{cdh}(\tau)$   
 $(\hat{e}, q, G_1, G_2) \xleftarrow{R} \mathbf{BDH}(\tau); g, h \xleftarrow{R} G_1 \setminus \{1\}; x \xleftarrow{R} \mathbb{Z}_q; a \leftarrow g^x$   
 $c \leftarrow \mathcal{A}(\hat{e}, q, G_1, G_2, g, a, h)$   
 If  $c = h^x$  return 1 else return 0

We let

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

denote the advantage of  $\mathcal{A}$  on input  $\tau$ , 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  $\nu$  for  $\tau$  if  $\mathbf{Adv}_{\mathcal{A}}^{cdh}(\tau) \leq \nu(\tau)$

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

**Definition 5.1.** We say that algorithm  $\mathcal{A}(\tau, t, \epsilon)$ -breaks the BDH parameter generator  $\mathbf{BDH}$  if  $\mathcal{A}$  runs for time at most  $t$ , and  $\mathbf{Adv}_{\mathcal{A}}^{cdh}(\tau)$  is at least  $\epsilon$ .

## 5.2 Chain Signature Protocol

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

**System Parameters** A Trusted Authority (TA) sets the parameter  $\tau$  and generates  $(\hat{e}, G_1, G_2, q) \xleftarrow{R} \mathbf{BDH}(\tau)$ . 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  $(e, q, G_1, G_2, \mathcal{H}, g)$ .

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

**ChainSign** Let  $L_i = \langle y_1, y_2, \dots, y_i \rangle$  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  $(\sigma_i, L_i)$ .

**ChainVerify** We accept the signature  $(\sigma_i, L_i)$  on  $m$  as valid if the following equality holds:

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

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

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

## 5.3 Security Of The Construction

We will reuse the proof of security of the short signature scheme of Boneh, Lynn and Shacham [14] (hereafter called BLS). The scheme is defined as follows.

**Preliminary Setup For BLS Signatures** A Trusted Authority (TA) sets a security parameter  $\tau$  and generates  $(\hat{e}, G_1, G_2, q) \xleftarrow{R} \text{BDH}(\tau)$ . 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 then selects a cryptographic hash function  $\mathcal{H} : \Sigma^* \mapsto G_1$  and picks a random generator  $g \in G_1$ . The system parameters are  $(\hat{e}, q, G_1, G_2, \mathcal{H}, g)$ .

**KeyGen** Generate  $x \xleftarrow{R} \mathbb{Z}_q^*$  as the private key. The public key is  $y = g^x \in G_1$ .

**Sign** To sign message  $m$  under public key  $y$ , compute  $h \leftarrow \mathcal{H}(m)$  and  $\sigma \leftarrow h^x \in G_1$ . Then  $\sigma$  is a valid BLS signature on message  $m$  under public key  $y$ .

**Verify** Accept the BLS signature  $\sigma$  under the public key  $y$  as valid if the following holds:

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

Security of BLS signatures is defined using Game 2.

### Game 2

**Setup.** The challenger sets the parameter  $\tau'$  and generates a key pair  $(x, y) \xleftarrow{R} \text{KeyGen}$ . It gives  $y$ , the challenge public key, along with  $\tau'$  to the adversary.

**Queries.** Working adaptively,  $\mathcal{A}$  issues at most  $q'_s$  sign queries and  $q'_h$  hash queries:

1. *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  $\sigma_i = \mathcal{H}(m_i)^x \in G_1$ .
2. *Hash queries:* For each hash query  $j$  on distinct messages  $m_h$  for  $1 \leq j \leq q'_h$ , the challenger responds with  $\mathcal{H}(m_j)$ .

**Output.** Finally  $\mathcal{A}$  outputs a BLS signature  $\sigma_A$ .

**Result:**  $\mathcal{A}$  wins if  $\sigma_A$  is a valid signature and no sign query was issued on  $m_A$ .

**Definition 5.2.** 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  $\tau'$ , 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  $\epsilon'$ . 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 [14].

**Theorem 5.3.** ([14, Theorem 3.2]) If there exists an algorithm  $\mathcal{A}$  that  $(\tau', t', q'_s, q'_h, \epsilon')$ -breaks the BLS scheme under Definition 5.2, then there exists another algorithm  $\mathcal{B}$  that  $(\tau'', t'', \epsilon'')$ -breaks the BDH parameter generator **BDH** under Definition 5.1, 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 5.4 below.

**Theorem 5.4.** Let there exist an algorithm  $\mathcal{A}$  that  $(n, \tau, t, q_s, q_e, q_h, e)$ -breaks the chain signature scheme under Definition 4.2. Then there exists another algorithm  $\mathcal{B}$  that  $(\tau', t', q'_s, q'_h, \epsilon')$ -breaks the BLS scheme under Definition 5.2, where;

$$\tau = \tau'; \quad t' \leq t + c_{G_1} \left[ n(q_h + 1) + q_s(n + 1) \right]; \quad q'_s \leq q_s; \quad q'_h \leq n(q_h + q_s); \quad \epsilon' \geq \frac{\epsilon}{e \cdot q_s}$$

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 a weak 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  $\tau'$  by challenger  $\mathcal{C}$ . Its goal is to forge a valid BLS signature under  $y$  using adversary  $\mathcal{A}$  that can win Game 1-a of Section 4.2.

Denote the BLS hash and sign oracles by  $\mathcal{H}_{\text{BLS}}$  and  $\text{Sign}_{\text{BLS}}$  respectively and denote the chain signature hash oracle by  $\mathcal{H}$ . The oracles  $\mathcal{H}_{\text{BLS}}$  and  $\text{Sign}_{\text{BLS}}$  will be simulated by  $\mathcal{C}$ , while  $\mathcal{H}$  will be simulated by  $\mathcal{B}$ .

**Setup.** Algorithm  $\mathcal{B}$  simulates the challenger of Game 1-a to adversary  $\mathcal{A}$ . It gives the parameter  $\tau = \tau'$  to  $\mathcal{A}$ , who returns an  $n$  bit value  $extr$ , indicating by 1 the private keys it wants to extract.

Denote the  $i^{\text{th}}$  bit of  $extr$  by  $bit_i$ . 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.

$\mathcal{B}$  generates  $r_i \xleftarrow{R} \mathbb{Z}_q$  and a bit  $a_i \xleftarrow{R} \{0, 1\}$  using a biased coin such that  $\Pr[a_i = 0] = 1/q_s$ . It then sets  $a_i \leftarrow a_i \cdot (1 - bit_i)$ , computes  $y_i = y^{a_i} g^{r_i} \in G_1$  and adds the entry  $(a_i, r_i, y_i)$  to the K-List.  $\mathcal{B}$  gives the set  $Y = \{y_i | 1 \leq i \leq n\}$  of challenge public keys, along with  $X = \{r_i \cdot bit_i | 1 \leq i \leq n\}$ , the set of extracted private keys to  $\mathcal{A}$ .

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

**Hash:** At any time  $\mathcal{A}$  may query the random oracle  $\mathcal{H}$ . To respond to hash queries,  $\mathcal{B}$  maintains another table called the H-List (which is initially empty and can have up to  $n(q_h + q_s)$  entries). Each entry  $i$  in the list is of the form,

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

and can be interpreted using Table 1.

Entry $i$	$b_i = 0$	$b_i = 1$
$u_i = 0$	$h_i = \mathcal{H}_{\text{BLS}}(m_i, L_i)$ $\gamma_i = \text{chain signature on } (m_i, L_i)$	$h_i = \mathcal{H}_{\text{BLS}}(m_i, L_i)$ $\gamma_i = \text{chain signature on } (m_i, L_i)/h_j^x$ for some $(m_j, L_j, b_j, u_j, h_j, \gamma_j) \in \text{H-List}$ 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 $(m_j, L_j, b_j, u_j, h_j, \gamma_j) \in \text{H-List}$ such that $m_i = m_j \wedge L_j \prec L_i \wedge b_j = 1 \wedge u_j > 0$ $\gamma_i = \text{chain signature on } (m_i, L_i)$	$h_i = \mathcal{H}_{\text{BLS}}(m_i, L_i)$ $\gamma_i = \text{chain signature on } (m_i, L_i)/h_i^x$
$u_i > 1$		$\text{Sign}_{\text{BLS}}$ 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  $\langle y_{j(1)}, y_{j(2)}, \dots, y_{j(|L_j|)} \rangle$  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  $L'_j = \langle y_{j(1)}, y_{j(2)}, \dots, y_{j(|L_j|-1)} \rangle$  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  $(m_j, L'_j)$  if  $|L_j| > 1$ .

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.<sup>3</sup>

<sup>3</sup>If  $|L_j| = 0$ ,  $\mathcal{B}$  simply replies with  $h_j = \mathcal{H}_{\text{BLS}}(m_j)$  without storing the value in the H-List.  $\mathcal{B}$ 's replies to  $\mathcal{A}$ 's hash

IF		$ L_j  > 1$	$ L_j  = 1$
$a_l = 0$		$h_j \leftarrow \mathcal{H}_{\text{BLS}}(m_j, L_j)$ $b_j \leftarrow b'_j; u_j \leftarrow 0$ $\gamma_j \leftarrow \gamma'_j \cdot h_j^{r_i}$	$h_j \leftarrow \mathcal{H}_{\text{BLS}}(m_j, L_j)$ $b_j \leftarrow 0; u_j \leftarrow 0$ $\gamma_j \leftarrow h_j^{r_i}$
$a_l = 1$	$b'_j = 0$	$h_j \leftarrow \mathcal{H}_{\text{BLS}}(m_j, L_j)$ $b_j \leftarrow 1; u_j \leftarrow 1$ $\gamma_j = \gamma'_j \cdot h_j^{r_i}$	$h_j \leftarrow \mathcal{H}_{\text{BLS}}(m_j, L_j)$ $b_j \leftarrow 1; u_j \leftarrow 1$ $\gamma_j \leftarrow h_j^{r_i}$
	$b'_j = 1$	$b_j \leftarrow 0; u_j \xleftarrow{R} \mathbb{Z}_q^*$ $h_j \leftarrow g^{u_j} / h_k$ , where $(m_k, L_k, b_k, u_k, h_k, \gamma_k) \in \text{H-List}$ 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 <b>largest</b> such sequence $\gamma_j \leftarrow \gamma'_j \cdot y^{u_j} \cdot h_j^{r_i}$	

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  $(m_k, L_k, b_k, u_k, h_k, \gamma_k) \in \text{H-List}$  such that  $(m_k = m_{s(i)}) \wedge (L_k \prec L_{s(i)}) \wedge (b_k = 1) \wedge (u_k > 0) \wedge (L_k \text{ is the largest sequence})$ .<sup>4</sup>
  - 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  $\sigma_{s(i)} \leftarrow \gamma_j \cdot \text{Sign}_{\text{BLS}}(m_k, L_k)$  by making a sign query to  $\mathcal{C}$  and returns  $\sigma_{s(i)}$  as its response to the chain sign query. Note that  $\text{Sign}_{\text{BLS}}(m_k, L_k) = h_k^x$
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  $(m_{s(i)}, L_{s(i)})$ .  $\mathcal{B}$  also keeps track of all such queried messages.

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

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-a, Algorithm  $\mathcal{B}$  reports Failure and terminates.

We know that  $(m_A, (\sigma_A, L_A))$  is a winning configuration of Game 1-a. 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. We know that  $b_A = 1$ .  $\mathcal{B}$  finds the entry  $(m_k, L_k, b_k, u_k, h_k, \gamma_k) \in \text{H-List}$  such that  $(m_k = m_A) \wedge (L_k \prec L_A) \wedge (b_k = 1) \wedge (u_k > 0) \wedge (L_k \text{ is the largest sequence})$ .<sup>5</sup>  
If  $u_k > 1$ ,  $\mathcal{B}$  reports Failure and terminates.

We know that

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

queries are indistinguishable from those of a random oracle. Therefore, the hash simulation provided by  $\mathcal{B}$  is perfect.

<sup>4</sup>It is possible that  $j = k$ . By analyzing Table 2, we can conclude that such an entry must necessarily exist.

<sup>5</sup>It is possible that  $L_k = L_A$ . By analyzing Table 2, we can conclude that such an entry must necessarily exist.

Therefore, by definition  $\gamma_A = \sigma_A/h_k^x$ , where  $h_k = \mathcal{H}_{\text{BLS}}(m_k, L_k)$ . In other words,  $\sigma_A/\gamma_A$  is a valid BLS signature under public key  $y$  on the message  $(m_k, L_k)$ . Additionally,  $u_k = 1$  implies that  $\mathcal{B}$  did **not** make a BLS sign query to  $\mathcal{C}$  on the message  $(m_k, L_k)$ .<sup>6</sup>

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

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

- $\mathcal{E}_1$ :  $\mathcal{A}$  wins Game 1-a.
- $\mathcal{E}_2$ : Event  $\mathcal{E}_1$  and  $b_A = 1 \wedge u_k = 1$  in Equation 1.

$\mathcal{B}$  succeeds if both these events happen. Therefore,  $\epsilon' = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2|\mathcal{E}_1]$ .

The following three claims give the bound on  $\epsilon'$ .

**Claim 1.**  $\Pr[\mathcal{E}_1] \geq \epsilon$

*Proof.* Since the simulation provided by  $\mathcal{B}$  is indistinguishable from a real game, the probability of  $\mathcal{A}$  winning the simulated Game 1-a is the same as the probability of  $\mathcal{A}$  winning Game 1-a. Thus,  $\Pr[\mathcal{E}_2|\mathcal{E}_1] \geq \epsilon$ .  $\square$

**Claim 2.**  $\Pr[\mathcal{E}_2|\mathcal{E}_1] = 1/(e \cdot q_s)$

*Proof.* Let  $q_s > 1$ . Using Equation 1, define events:

- $\mathcal{E}_3$ :  $b_A = 1$
- $\mathcal{E}_4$ : Event  $\mathcal{E}_3$  and  $u_k = 1$

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

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  $y_l$ . For any sequence  $L_* \in L$  or set  $L_* \in 2^Y$ , define

$$\text{Sum}(L_*) = \sum_{y_l \in L_*} a_l$$

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

**Result:**  $\mathcal{A}$  wins Game 1-a if all the following conditions hold:

- (a) (a)  $L_A \in L$  and the **ChainVerify** algorithm accepts  $(\sigma_A, (m_A, L_A))$  as valid.
- (b) No chain sign query has been previously made on the pair  $(m_A, L_A)$ .
- (c) At least one private key corresponding to  $L_A$  has not been extracted.
- (b) For **each** chain sign query  $i$ , if  $(m_{s(i)} = m_A) \wedge (\{L_A, L_{s(i)}\} \text{ overlap})$ , then there is at least one key in  $(L_{s(i)} \cup L_A) \setminus (L_{s(i)} \odot L_A)$  which has not been extracted.

By checking Table 2, we can verify that

$$\begin{aligned} \mathcal{E}_3 &\Rightarrow \text{Sum}(L_A) \equiv 1 \pmod{2} \\ \mathcal{E}_4 &\Rightarrow (\text{Sum}(L_{s(i)} \setminus L_k) < 1) \wedge (m_A = m_{s(i)}) \wedge (L_k \prec L_{s(i)}) = \text{False} \quad \forall i \ (1 \leq i \leq q_s) \end{aligned}$$

For event  $\mathcal{E}_3$ , we require that of the unextracted keys in  $L_A$ , an odd number of keys  $y_l$  have  $a_l = 1$ . Thus,  $1 - 1/q_s \geq \Pr[\mathcal{E}_3|\mathcal{E}_1] \geq 1/q_s$ .

Consider the following three cases:

<sup>6</sup>To visualize this, consider the example of Section 4.5 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”. In this instance of the game  $L_k = \langle y_1, y_2, y_3, y_4 \rangle$ . Observe that  $\mathcal{B}$  would never have made a  $\text{Sign}_{\text{BLS}}$  query to  $\mathcal{C}$  on the message  $(m_A, \langle y_1, y_2, y_3, y_4 \rangle)$ .

- **Case 1:**  $\mathcal{A}$  did not make any non-trivial chain sign queries. In this case  $u_k = 1$  (necessarily), and  $\Pr[\mathcal{E}_4 | \mathcal{E}_3 \wedge \mathcal{E}_1] = 1$ .
- **Case 2:**  $\mathcal{A}$  made sign queries  $i$  such that  $(m_A = m_{s(i)}) \wedge (L_A \odot L_{s(i)} \neq \emptyset)$  but for each such query  $i$ , at least one key (say  $y_l$ ) in  $L_A \setminus L_A \odot L_{s(i)}$  is unextracted. In the worst case, this unextracted key in  $L_A$  would be the same for all such queries, implying that all other keys in  $L_A$  are extracted. Then with probability  $1 - 1/q_s$ , we would have  $a_l = 1$ , in which case  $u_k = 1$ , and so we have

$$\Pr[\mathcal{E}_4 | \mathcal{E}_3 \wedge \mathcal{E}_1] \geq 1 - \frac{1}{q_s}$$

- **Case 3:** The only case left to consider is when one or more chain sign queries  $i$  have been made such that

$$(m_A = m_{s(i)}) \wedge (L_A \odot L_{s(i)} \neq \emptyset) \wedge (\text{All keys in } L_A \setminus (L_A \odot L_{s(i)}) \text{ are extracted})$$

Let us denote this set of queries by  $Q$ . In the worst case, all keys except the ones restricted by the winning condition are extracted. This implies that for each such query  $i \in Q$ , there is at least one key in  $L_{s(i)} \setminus (L_A \odot L_{s(i)})$  that has **not** been extracted. Now consider the set of queries  $Q_* \subseteq Q$  defined as

$$Q_* = \{i | (i \in Q) \wedge (L_k \prec L_{s(i)})\}$$

Then for each  $i \in Q_*$ , there is (necessarily) at least one key (say  $y_l$ ) in  $L_{s(i)} \setminus L_k$  that has **not** been extracted. Event  $\mathcal{E}_4$  implies that for each of these  $|Q_*|$  unextracted keys  $y_l^*$ , the corresponding bits  $a_l^*$  in the K-List are 1.

From the simulation, it is clear that the values of the above bits  $a_l^*$  are independent of  $\mathcal{A}$ 's view and so  $\Pr[a_l^* = 1] = 1 - 1/q_s$  independent of the other bits. Since  $|Q_*| \leq q_s$ , we have,

$$\Pr[\mathcal{E}_4 | \mathcal{E}_3 \wedge \mathcal{E}_1] = \Pr[\text{At most } q_s \text{ independent bits } a_l^* \text{ are 1}] \geq \left(1 - \frac{1}{q_s}\right)^{q_s} \geq \frac{1}{e}$$

$$\therefore \Pr[\mathcal{E}_2 | \mathcal{E}_1] = \Pr[\mathcal{E}_4 | \mathcal{E}_3 \wedge \mathcal{E}_1] \cdot \Pr[\mathcal{E}_3 | \mathcal{E}_1] \geq \frac{1}{e \cdot q_s}$$

□

Combining Claims 1 and 2, we get the bound on  $\epsilon'$ .

**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 generate up to  $n$  public keys; the time required to add 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 generating a public key 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  $n$  public keys, 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 5.4. □

The above proof is similar to the proof of security of Verifiably Encrypted Signatures (VES) of [3]. In fact, the security of CS in the weak known key attack model is very similar to the security of VES against signature extraction and forgery. It is possible that our construction is also secure in the stronger sense of adaptive known key and chosen message attacks given by Definition 4.1. In fact, we posit that an efficient reduction exists from an adversary of Game 1 to an adversary of Game 1-a.

## 5.4 Adaptive Security In the Chosen Key Model

We note that above construction of chain signatures is possibly 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). Our intuition behind this claim is the following: Any chain signature  $(\sigma_*, (m_*, L_*))$  is also an **aggregate** signature on some 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.

## 5.5 Identity Based Chain Signatures (IBCS)

The protocol presented in Section 5.2 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 [18, 19] support signature aggregation with the property that once aggregated, individual signatures cannot be extracted. The security model for IBCS under adaptive chosen key and chosen message attacks would be identical to that of CS under adaptive known key and chosen message attacks described in Game 1. Unlike standard signatures, however, no known constructions of IBS are known where the size of the aggregate signature is constant. This is an interesting open problem.

## 5.6 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.<sup>7</sup> 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 existentially unforgeable signature scheme. This is the approach we follow in Section 6.1.1.

## 5.7 Efficiency Of Chain Signatures

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 [20] indicate that each pairing operation using these parameters takes  $\approx 8.6$ ms 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 [20]. Using these values and neglecting the faster operations, we obtain the following performance estimates of the above protocol (assuming  $n$  users in the chain):

<sup>7</sup>To see why the security reduction is complicated, the reader is encouraged to try and prove the security for distinct messages using the above-mentioned approach. The problem arises because with different messages, the proof technique of Theorem 5.4 does not work - the probability of success after the adversary outputs a valid forgery becomes negligibly small if the individual signatures are not “linked” using the same message.



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

## 6 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 [21, 22], group e-commerce, work-flow enforcement, secure routing, authenticated mail relaying, Grid computing and Mobile IP. The applications to mobile agent systems have already been discussed in Section 2. Here, we discuss another application: stateless routing.

### 6.1 Stateless Routing

The most common and robust interior and exterior routing protocol is the Border Gateway Protocol (BGP) [23, 24]. It is a **Path Vector Routing** protocol, in which 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 route to all its other immediate neighbors. Although BGP is very robust, it has many security vulnerabilities [25, 26]. For instance, a rogue router could send forged updates or extract intermediate routes from legitimate updates and claim to a shorter route (we call the latter a *Path Extraction Attack*). This is best explained using an example.

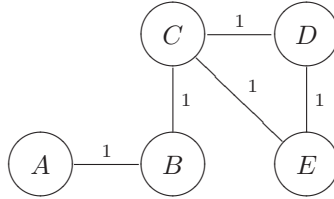


Figure 3: Scenario for a Path Extraction Attack

Consider the network of routers given in Figure 3. 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. 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) [27] 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

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

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 [10, 3, 28]. For instance, aggregate signatures have been proposed to keep the signature payload to a constant size [3]. The authors of [27] propose the use of *Signature Amortization* [28] coupled with aggregate or sequential aggregate signatures [6] to reduce the size of update messages and the signing time. However, all the above works assume some sort of stateful environment, where information about peers 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 [27].

### 6.1.1 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.

Denote by  $y_i$  the public key of user  $i$  under a chain signature scheme. Let  $Sign_i, Verify_i$  denote sign and verify functions of user  $i$  under another existentially unforgeable signature scheme, such as BLS. Denote by  $L_i = \langle y_1, y_2, \dots, y_n \rangle$  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. 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  $M_i$ . 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  $Sig_1 = \text{Sign}_1(M, M_1)$  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.  $\text{Verify}_i(Sig_i, (M, M_i)) = \text{True}$ .
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  $\sigma_i$  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  $Sig_{i+1} = \text{Sign}_{i+1}(M, M_{i+1})$ .
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 Section 5.5) 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  $(\sigma_i, L_i)$ . 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].<sup>8</sup>

---

<sup>8</sup>The use of aggregate signatures in BGP advertisement verification has already been discussed in [3]. We additionally suggest using chain signatures for increased scalability.

## 7 Summary

In this paper, we introduced the notion of Chain Signatures as an extension of Boneh et al.'s short signatures [14]. 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 4.2 and 4.5. We note that chain signatures without using bilinear maps were independently proposed in [22, 29] in which the authors used hypothetical primitives called *Strong Associative One-Way Functions* (SAOWFs).

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 [29, 22], 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 *truncation resilience* (See Section 3). The chain signature scheme presented here, however, does not provide *strong truncation resilience*. Signatures that also provide strong truncation resilience 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] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In Thomas Johansson and Subhamoy Maitra, editors, *INDOCRYPT*, volume 2904 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2003.
- [5] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485. Springer, 2006.
- [6] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2004.
- [7] 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.
- [8] Yon Dohn Chung, Jong Wook Kim, and Myoung-Ho Kim. Efficient preprocessing of xml queries using structured signatures. *Inf. Process. Lett.*, 87(5):257–264, 2003.
- [9] Chih-Yin Lin, Tzong-Chen Wu, and Jing-Jang Hwang. Id-based structured multisignature schemes. In *Proceedings of the IFIP TC11 WG11.4 First Annual Working Conference on Network Security*, pages 45–60, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

- [10] Huafei Zhu, Feng Bao, Tieyan Li, and Yongdong Wu. Sequential aggregate signatures for wireless routing protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, volume 4, pages 2436–2439. IEEE Computer Press, March 2005.
- [11] 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.
- [12] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the computation results of free-roaming agents. In *MA '98: Proceedings of the Second International Workshop on Mobile Agents*, pages 195–207, London, UK, 1999. Springer-Verlag.
- [13] 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.
- [14] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptology*, 17(4):297–319, 2004.
- [15] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.
- [16] 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.
- [17] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.
- [18] 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.
- [19] B. Libert and J. Quisquater. The exact security of an identity based signature and its applications. Technical Report 2004/102, Cryptology ePrint Archive, 2004.
- [20] 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.
- [21] Amitabh Saxena and Ben Soh. A mobile agent authentication protocol using signature chaining with bilinear pairings. Cryptology ePrint Archive, Report 2005/272, 2005.
- [22] 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.
- [23] Y. Rekhter, T. Li, and S.Hares. RFC 4271: A Border Gateway Protocol 4 (BGP-4), jan 2006.
- [24] Jennifer Rexford, Jia Wang, Zhen Xiao, and Yin Zhang. Bgp routing stability of popular destinations. In *ACM SIGCOMM IMW (Internet Measurement Workshop) 2002*, 2002.
- [25] S. Murphy. RFC 4272: BGP Security Vulnerabilities Analysis, January 2006.
- [26] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding bgp misconfiguration. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–16, New York, NY, USA, 2002. ACM Press.

- [27] 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.
- [28] 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.
- [29] 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.