Bounded Vector Signatures and their Applications

Lei Wei
University of North Carolina
Chapel Hill, NC, USA
lwei@cs.unc.edu

Scott E. Coull
RedJack, LLC.
Silver Spring, MD, USA
scott.coull@redjack.com

Michael K. Reiter
University of North Carolina
Chapel Hill, NC, USA
reiter@cs.unc.edu

ABSTRACT

Although malleability is undesirable in traditional digital signatures, schemes with limited malleability properties enable interesting functionalities that may be impossible to obtain otherwise (*e.g.,* homomorphic signatures). In this paper, we introduce a new malleable signature scheme called *bounded vector signatures*. The proposed scheme allows a user to sign a multi-dimensional vector of values, along with a description of the context within which the vector should be interpreted. The scheme includes a unique malleability property, which we refer to as the *stretch property*, that allows the components of the signed vector to be increased up to a pre-defined limit without access to the signing key. Decreasing these values, however, remains computationally infeasible. We prove the security of our construction under the strong RSA and decisional Diffie-Hellman assumptions in the random oracle model. Finally, we underscore the utility of bounded vector signatures by discussing their use in distributed systems security applications.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

Algebraic Signatures, Malleable Signatures, Vectors

1. INTRODUCTION

Typically, digital signature schemes must remain secure against existential forgeries to achieve their purpose of verifying the authenticity of data. Therefore, it would seem as though any malleability allowed by a signature scheme would be strictly harmful. However, several malleable signature schemes, including homomorphic signatures, have been proposed to achieve functionalities by allowing arbitrary parties to perform a limited set of operations to alter the signed data in logical ways. Transitive signatures [32, 6, 33, 23], for instance, allow a user with signatures on graph edges (x, y) and (y, z) to produce a third signature on the transitive closure, (x, z), without access to the signing key.

Here, we propose a new type of malleable signature, called *bounded vector signatures*. Intuitively, our scheme allows a user to sign a vector of natural numbers, and includes a unique malleability property that allows components of these signed vectors to be increased, or *stretched*, up to a pre-defined bound without access to the signing key. Furthermore, each vector is associated with an arbitrary string, which we refer to as the *context*, that describes the way to interpret the values within the signed vector. To capture the security of our scheme, we define *shrink unforgeability*, which captures the inability of a computationally bounded adversary to decrease the value of any of the signed vector components.

The malleability property of our bounded vector signature scheme is particularly useful in multiparty settings, which are the primary focus of this paper. In such a setting, n signers may produce partial signatures on vectors, and any $t \leq n$ of those partial signatures can be efficiently combined to form a full signature on a vector representing all signers. In a typical, non-malleable signature scheme, these partially signed vectors would have to be exactly the same for the combiner to create a full signature. The malleability property of our bounded vector signatures, however, allows the signers to sign *different* vectors. In particular, the combiner receives t partial signatures on different vectors, applies the stretch operation to those signed vectors until all of them are the same, and then combines them as before.

This form of malleability leads to very efficient solutions to many problems that arise in the area of distributed systems security. In these distributed systems problems, information must be securely aggregated from multiple (potentially untrusted) parties while simultaneously limiting communications overhead and interaction among the parties. To that end, we show how our bounded vector signatures can be used to sign a number of interesting data structures commonly used in distributed environments, such as sets and intervals, with just a *small constant number* of elements (*e.g.,* at most four) of the underlying multiplicative group. Moreover, for each of these data structures we describe how the malleability of our bounded vector signatures enables us to *non-interactively* compute a full signature on exactly the intersection or union of given partially signed sets or intervals. We also discuss the application of these methods to distributed systems problems, including distributed cache reconciliation in content delivery networks [19, 9, 35], fault tolerant computation in sensor networks [31], and group key management [34, 41].

Related Work Malleable signatures, which can include homomorphic signature schemes and are sometimes referred to as algebraic signatures, were first discussed by Rivest during a series of talks at Cambridge University. The concept was later formalized by Johnson *et al.* [26], who described homomorphic signature schemes for redacted documents and set operations. Since then, new homomorphic signature schemes have been introduced, including transitive signature schemes on graphs [23, 6, 32, 33] and signatures for network coding [8, 22, 1]. Additionally, Kiltz *et al.* [28] introduced a malleable signature scheme that allows a user to append a message to a signature without requiring a signature on the appended message itself. Our bounded vector signatures, like the append-only signatures of Kiltz *et al.*, do not require a specific homomorphic property of the signature and instead rely upon a limited form of malleability. We note that the functionality allowed by our vector signatures is orthogonal to that provided by Kiltz *et al.*

The limited malleability allowed by our bounded vector signatures is achieved through an RSA-like signature construction, where the components of the vector being signed are encoded in the exponent of the signature, rather than the base. The first such construction was proposed by De Jonge and Chaum [16] in a study of several RSA signature variations. Gennaro *et al.* [21], and Hohenberger and Waters [25] have since used the construction, along with division-intractable and chameleon hashes, to build hash-and-sign signatures without random oracles. Perhaps the most closely related construction to our own is the dynamic accumulator scheme of Camenisch and Lysyanskaya [10], where identities are encoded as random primes in the exponent of the accumulator. None of these constructions, however, were developed with the goal of malleability in the form we pursue here, nor were they developed for use in multiparty settings typical of distributed systems problems.

2. PRELIMINARIES

To begin, we introduce bounded vector signatures, and define a new notion of unforgeability, called shrink unforgeability. Throughout the paper, we denote a d-dimensional vector of natural numbers as $\mathbf{v} =< v_1, \ldots, v_d >$, where the k^{th} component is denoted as $\mathbf{v}[k]$. Scalar values are denoted by non-bolded variables. Furthermore, we use the symbol $\overset{\$}{\leftarrow}$ to indicate sampling uniformly at random from a set of values, and the symbol $\leftarrow$ to indicate the output of an algorithm.

2.1 Bounded Vector Signatures

A bounded vector signature scheme is a malleable signature on d-dimensional vectors wherein the components of the signed vector may be altered, within a predetermined bound, without access to the associated signing key. Specifically, we consider a signature on a pair $(\mathbf{v}, c)$, where $\mathbf{v} \in \mathbb{N}^d$ is a d-dimensional vector in the natural numbers and c is a context (*i.e.,* a text description) within which the vector is interpreted. As an example, $\mathbf{v}$ may represent distances in d independent dimensions and c may describe the starting point of the vector within that space. Furthermore, bounded vector signatures have a unique property that allows arbitrary parties to increase the values of any component of the signed vector up to a predefined limit without access to the signing key. For the remainder of this paper, we refer to this property as the *stretch property.*

Since we are most interested in distributed applications, we present a multiparty version of our scheme. The multiparty bounded vector signature scheme follows the typical threshold signature paradigm wherein n signers, which we denote as $P_1, \ldots, P_n$, may produce partial signatures on vector-context pairs. Any subset of t distinct partial signatures can then be combined to produce a full signature that can be verified using a single verification key. Both the partial and full signatures in the multiparty version of our bounded vector signature scheme inherit the stretch property. Therefore, it is possible to combine partial signatures on different vectors by stretching them until they are identical, as long as their contexts are the same. In particular, the resultant vector in the full signature will be equal to (at least) the component-wise maximum from among all t vector signatures. For ease of exposition, we present the formal description of our scheme and its instantiation in a multiparty setting, and note that the single signer case may be trivially instantiated by setting $n = t = 1$.

More formally, a *bounded vector signature scheme* is defined by a tuple of algorithms $\mathcal{BVS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Stretch}, \mathsf{Combine}, \mathsf{Verify})$ run by n signers $P_1, \ldots, P_n$. The input-output specifications of these algorithms are as follows:

$\mathsf{KeyGen}(1^\kappa, t, n, \hat{\mathbf{v}})$: The key generation function takes as input a security parameter 1^κ, the threshold number t of partial signatures necessary to form a final signature, the number of signers n, and a d-dimensional vector $\hat{\mathbf{v}}$ containing the maximum value (*i.e.,* bound) for each dimension. The algorithm outputs signing keys $sk_1, \ldots, sk_n$ for each of the signers $P_1, \ldots, P_n$, and a single verification key pk used to verify the final signatures.

$\mathsf{Sign}(sk_i, (\mathbf{v}, c))$: The signing algorithm takes in a signing key sk_i, along with a vector-context pair $(\mathbf{v}, c)$, where $0 \leq \mathbf{v}[k] \leq \hat{\mathbf{v}}[k]$ for each dimension k. The context describes how the verifier should interpret the signed value, such as the starting point of the vector. The signing algorithm outputs a partial signature on vector $\mathbf{v}$ associated with context c for signer P_i, which we denote as σ_i.

$\mathsf{Stretch}((\sigma, \mathbf{v}, c), k, a)$: The stretch algorithm takes a partial or full signature σ on vector-context pair $(\mathbf{v}, c)$ and an amount a to increment the value of the k^{th} component. The output of the algorithm is a new (partial or full) signature σ' on $(\mathbf{v}', c)$, where $\mathbf{v}'[k] = min(\mathbf{v}[k] + a, \hat{\mathbf{v}}[k])$.

$\mathsf{Combine}((\sigma_{i_1}, \mathbf{v}_{i_1}, c_{i_1}), \ldots, (\sigma_{i_t}, \mathbf{v}_{i_t}, c_{i_t}))$: The combine algorithm takes as input a set of t partial signatures on vector context pairs $(\mathbf{v}_{i_j}, c_{i_j})$ from distinct signers. These signatures must be associated with the same context c (*i.e.,* $c_{i_j} = c$), but may sign t different vectors. The algorithm outputs a full signature σ on the component-wise maximum values among the input signatures and their associated context (*i.e.,* $(\mathbf{v}', c)$), where $\mathbf{v}'[k] = max(\mathbf{v}_{i_1}[k], \ldots, \mathbf{v}_{i_t}[k])$ for all dimensions k.

$\mathsf{Verify}(pk, \sigma, (\mathbf{v}', c'))$: The verify algorithm takes in the verification key pk, a full signature σ, and a vector-context pair $(\mathbf{v}', c')$ that is to be verified as being signed. If the signature is valid for the vector-context pair, then the algorithm outputs 1. Otherwise, the algorithm outputs 0.

2.2 Security for Bounded Vector Signatures

Due to the malleable nature of our proposed signature scheme, the traditional definition of existential unforgeability under chosen message attack does not appropriately capture its security. Instead, we propose the notion of *shrink unforgeability* under chosen message attack. This definition captures the ability of untrusted parties to *increase*, but not decrease, the values of any component of the signed vector without access to the signing key. In the multiparty setting, this restriction translates to a limitation on the values within each component of the fully signed vector output from the Combine procedure. Specifically, the adversary should not be able to produce a full signature on a vector where the value in the k^{th} dimension is less than the t^{th} smallest value in that dimension from among all partial signatures given as input to the Combine algorithm. The intuition here is simply that the adversary would have to *decrease* at least one value in order to create a full signature on a vector containing values less than the t^{th} smallest in each dimension, which is disallowed by shrink unforgeability. This security notion is formalized as follows.

DEFINITION 1 (SHRINK UNFORGEABILITY). *Let $\mathcal{BVS}$ be a bounded vector signature scheme, and let $\mathcal{A}$ be an algorithm with oracle access to n signing oracles representing the signers $P_1, \ldots, P_n$. Each query $(\mathbf{v}, c)$ to oracle $\textbf{\textit{Oracle}}_{sk_i}$ returns $\mathsf{Sign}(sk_i, (\mathbf{v}, c))$. For any context c and dimension k, let $\mathbf{v}_1^c[k], \ldots, \mathbf{v}_n^c[k]$ denote the minimum values queried by $\mathcal{A}$ with context c to the n oracles, listed in nondecreasing order. If $\mathcal{A}$ made queries on context c to only $s < n$ oracles, then we let $\mathbf{v}_{s+1}^c[k] = \cdots = \mathbf{v}_n^c[k] = \hat{\mathbf{v}}[k] + 1$. The experiment for shrink unforgeability under chosen message attack is then defined as follows:*

```
Experiment $\mathbf{Exp}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A})$:
    $(pk, sk_1, \ldots, sk_n) \leftarrow \mathsf{KeyGen}(1^\kappa, t, n, \hat{\mathbf{v}})$
    $(\mathbf{v}', c', \sigma') \leftarrow \mathcal{A}^{\textbf{\textit{Oracle}}_{sk_1}, \ldots, \textbf{\textit{Oracle}}_{sk_n}}(pk)$
    Return 1 if $\mathsf{Verify}(pk, \sigma', (\mathbf{v}', c')) = 1$ and either:
        (1) $\mathcal{A}$ queried $(*, c')$ to less than $t$ signing oracles, or
        (2) $\mathcal{A}$ queried $(*, c')$ to at least $t$ signing oracles and
            for some dimension $k$, $\mathbf{v}'[k] < \mathbf{v}_t^{c'}[k]$.
```

Let $\mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A}) = \mathbf{Pr}\left[\mathbf{Exp}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A}) = 1\right]$, and define $\mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(T) = \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A})$ where the maximum is taken over all adversaries $\mathcal{A}$ running in time at most T.

We briefly explain the adversary's actions in this experiment. An adversary is provided with n partial signature oracles representing n signers. He can make partial signature queries on any $(\mathbf{v}, c)$ pairs he chooses in any order and run the Combine algorithm to get full signatures from the partial signatures he received. In the end, he successfully breaks the shrink unforgeability if he outputs a valid signature on a pair $(\mathbf{v}', c')$ and either of the two following cases occurred. The first case is when a forger $\mathcal{A}$ makes signature queries of the form $(*, c')$ to less than t signing oracles. In this case, he should not be able to generate full signatures on any pair of the form $(*, c')$, since he did not collect enough partial signatures. The second case is when $\mathcal{A}$ made signature queries of the form $(*, c')$ to at least t signing oracles and there exists a dimension k such that $\mathbf{v}'[k] < \mathbf{v}_t^{c'}[k]$. That is, $\mathbf{v}_t^{c'}[k]$ is the smallest value in dimension k on which a signature should be possible for context c'.

2.3 Strong RSA Assumption

Here, we briefly recall the definition of the strong RSA assumption, which was first proposed by Barić and Pfitzmann [5]. In this paper we are primarily interested in the following variant of the strong RSA problem where the input is chosen as a random quadratic residue. It has been shown that this variant is not any easier than the standard strong RSA problem [13].

Let N be a positive integer that is the product of two distinct, random κ-bit primes p, q, let QR_N denote the quadratic residues in $\mathbb{Z}_N^*$, and let y be a random element of QR_N. Informally, given a random (N, y), the Strong RSA assumption is that it is hard to compute a pair (x, e) such that $e > 1$ and $x^e \equiv y \mod N$. Specifically we denote the advantage of an adversary $\mathcal{B}$ in solving the strong RSA problem as

$$\begin{aligned}\mathbf{Adv}_{\mathsf{QR\text{-}SRSA}}(\mathcal{B}) = \Pr[\ & e > 1 \text{ and } x^e \equiv y \bmod N : \\ & p, q \xleftarrow{\$} \mathsf{Primes} \cap \{0,1\}^\kappa, N \leftarrow pq, \\ & y \xleftarrow{\$} QR_N, (x, e) \leftarrow \mathcal{B}(N, y) \quad]\end{aligned}$$

and we define $\mathbf{Adv}_{\mathsf{QR\text{-}SRSA}}(T) = \max_{\mathcal{B}} \mathbf{Adv}_{\mathsf{QR\text{-}SRSA}}(\mathcal{B})$ where the maximum is taken over all adversaries $\mathcal{B}$ executing in time at most T.

2.4 Decision Diffie-Hellman Assumption

Our proof of security also makes use of the decisional Diffie-Hellman (DDH) assumption in the quadratic residue group QR_N when N is the multiplication of two strong primes. That is, it is difficult for an adversary to distinguish a triple (g^a, g^b, g^{ab}) from (g^a, g^b, g^c) for a, b, and c chosen randomly and for a generator g of QR_N. In particular, we use a variant of the problem which says that DDH problem is hard even when the factorization of N is known. It has been shown that the DDH problem over QR_N does not depend on the hardness of factoring [27]. Formally, the DDH advantage of an adversary $\mathcal{D}$ is defined as follows, where SafePrimes are the primes p of the form $p = 2p' + 1$ where p' is itself a prime:

$$\begin{aligned}&\mathbf{Adv}_{\mathsf{DDH}}(\mathcal{D}) = \\ &|\Pr[\ \mathcal{D}(p, q, g, g^a, g^b, g^{ab}) = 1 : \\ &\quad p, q \xleftarrow{\$} \mathsf{SafePrimes} \cap \{0,1\}^\kappa, N \leftarrow pq, g \xleftarrow{\$} QR_N, \\ &\quad a, b \xleftarrow{\$} \mathbb{Z}_{p'q'} \text{ where } p' = (p-1)/2, q' = (q-1)/2 \quad] - \\ &\ \Pr[\ \mathcal{D}(p, q, g, g^a, g^b, g^c) = 1 : \\ &\quad p, q \xleftarrow{\$} \mathsf{SafePrimes} \cap \{0,1\}^\kappa, N \leftarrow pq, g \xleftarrow{\$} QR_N, \\ &\quad a, b, c \xleftarrow{\$} \mathbb{Z}_{p'q'} \text{ where } p' = (p-1)/2, q' = (q-1)/2\ ]\ |\end{aligned}$$

Let $\mathbf{Adv}_{\mathsf{DDH}}(T) = \max_{\mathcal{D}} \mathbf{Adv}_{\mathsf{DDH}}(\mathcal{D})$ where the maximum is taken over all adversaries $\mathcal{D}$ executing in time at most T.

Below, we refer to a triple (g^a, g^b, g^{ab}) as a *valid* DDH triple, and (g^a, g^b, g^c) as an *invalid* DDH triple.

3. OUR CONSTRUCTION

In this section, we present a bounded vector signature construction in the multiparty setting using the threshold RSA signature scheme of Shoup [38] as the basis of our construction. The novelty of our construction lies in the observation that values being signed can be moved from the base of the signature to the exponent. By doing so, we can encode each value in such a way that arbitrary parties can increase this value by exponentiating the signature, but cannot decrease the value without being able to take roots in the RSA group. In particular, we encode the signature on the value in a vector dimension as the distance from that value to the maximum value for that dimension, and then embed that distance as a root in the RSA group. Thus, by exponentiating the signature by an exponent corresponding to this dimension, the distance in that dimension can be decreased (and so the value in that dimension is increased). The context portion of the signature acts as the base of the RSA signature and provides the meaning behind the values in the exponent. The context also ensures that vectors in different contexts cannot be combined.

3.1 Construction Description

Here we discuss the operation of each algorithm in our construction and provide the technical details in Figure 1.

The signature scheme is initialized by running the KeyGen algorithm, which generates the public verification key and the secret signing keys for each of the n signers. For the multiparty setting, we consider a trusted dealer[1] who generates two safe primes $p = 2p' + 1, q = 2q' + 1$, and a RSA modulus $N = pq$. Each dimension k in the vector space being signed is associated with a maximum value $\hat{\mathbf{v}}[k]$ that is given as input to the algorithm and a distinct prime e_k chosen by the dealer. The dealer also computes the signing key $sk = \prod_{k=1}^{d} e_k^{-(\hat{\mathbf{v}}[k]+1)} \bmod m$, where $m = p'q'$. The dealer then splits sk into n shares $sk_1, \ldots, sk_n$ using standard polynomial secret sharing techniques [37]. Essentially, this signing key encodes the maximum possible distance for each dimension in the index.

The Sign algorithm allows a signer with signing key sk_i to produce a partial signature on a d-dimensional vector. The signer first creates a context c, which is a text description of the way the verifier should interpret the signed vector. Practically speaking, the format of the context is application specific, but its general purpose in our construction is to ensure that vectors from different contexts cannot be combined into a full signature. The output of the Sign algorithm is a partial signature $\sigma_i = \mathcal{H}(c)^{n! sk_i \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$, where $\mathcal{H}(\cdot)$ is a hash into the group QR_N. The inclusion of $n!$ in the exponent is taken from the work of Shoup [38] to ensure that the Lagrange interpolation performed during the $\mathsf{Combine}$ algorithm produces integers.

Given a partial or full signature on a vector $\mathbf{v}$, an arbitrary party (without access to any signing key) can run the $\mathsf{Stretch}$ algorithm to increase the value of any component of the vector up to the pre-determined maximum set by the vector $\hat{\mathbf{v}}$ in the KeyGen algorithm. To do so, the party exponentiates the signature by the prime associated with the desired dimension to increment the value in that dimension (*i.e.,* decreasing the distance from the maximum). The output of the algorithm is a new partial or full signature $\sigma' = (\sigma)^{e_k^a} \bmod N$, where k is the dimension being incremented and a is the amount by which it is increased.

The $\mathsf{Combine}$ algorithm takes t partial signatures on potentially different vectors from distinct signers and produces a full signature from them. In order to properly combine partial signatures on different vectors, the algorithm begins by applying the $\mathsf{Stretch}$ operation on each dimension of each partial signature to ensure all partial signatures provided as input sign the same vector. The resultant vector is actually the component-wise maximum among the t signed vectors given as input. Once all partial signatures embed the same vector, the $\mathsf{Combine}$ algorithm computes Lagrange coefficients for each partial signature and interpolates the signing key sk in the exponent. Thus, the output of the algorithm is a full signature $\sigma = \mathcal{H}(c)^{sk \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} = \mathcal{H}(c)^{\prod_{k=1}^{d} e_k^{-(\hat{\mathbf{v}}[k]-\mathbf{v}[k]+1)}} \bmod N$ on the vector $\mathbf{v}$. Notice that in addition to requiring all vectors to be stretched to be identical, the context of these vectors must also be the same for the $\mathsf{Combine}$ algorithm to successfully produce a full signature. It is also important to note that this $\mathsf{Combine}$ procedure, after stretching, is exactly that of Shoup [38], and so interested readers should refer to that work for detailed technical explanation of the procedure.

Finally, the Verify algorithm takes as input a full signature and a vector-context pair, and outputs 1 if the full signature embeds the given vector-context pair. The verification procedure simply checks that $\mathcal{H}(c) = \sigma^{\prod_{k=1}^{d} e_k^{\hat{\mathbf{v}}[k]-\mathbf{v}[k]+1}} \bmod N$, which only occurs if the signature encodes the given vector components as the appropriate distance from the predetermined maximum for each dimension.

We prove that this construction is shrink unforgeable under the strong RSA and DDH assumption in the random oracle model. Since we model $\mathcal{H}()$ as a random oracle, we additionally quantify the number of queries q_h made to $\mathcal{H}()$ in the adversary advantage. More formally, $\mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(T, q_h) = \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A})$ where the maximum is taken over all adversaries $\mathcal{A}$ running time at most T and making at most q_h queries to $\mathcal{H}()$.

THEOREM 1. *Consider the above bounded vector signature construction with d dimensions and each vector dimension restricted to the range $[0, \hat{v}]$. Then, ignoring terms negligible in κ,*

$$\begin{aligned}
&\mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(T, q_h) \\
&\quad \leq \; 2 \cdot q_h \cdot d \cdot \hat{v} \cdot \left(\frac{n}{t-1}\right)^{2(t-1)} \cdot \mathbf{Adv}_{\mathsf{QR\text{-}SRSA}}(T') \\
&\qquad + \; d \cdot \hat{v} \cdot \left(\frac{n}{t-1}\right)^{2(t-1)} \cdot (t-1) \cdot \mathbf{Adv}_{\mathsf{DDH}}(T'')
\end{aligned}$$

where $T' = T'' = T + O(t(d\hat{v}\log(n + d\log n) + \log(n!)))$ group operations.

The proof of the theorem can be found in Appendix A.

3.2 Efficiency

With respect to space efficiency, the signing keys consist of a group element in $\mathbb{Z}_m$, while the public key is made up of an RSA modulus and d prime exponents. Both the partial and full signatures in our construction are represented by a single group element in QR_N.

[1]Existing techniques for removing the assumption of a trusted dealer may be applied to our construction (*e.g.,* [15]).

```
KeyGen$(1^\kappa, t, n, \hat{\mathbf{v}})$:
    1. Dealer selects random $\kappa$-bit primes $p = 2p' + 1$, $q = 2q' + 1$ where $p'$ and $q'$ are themselves prime. Dealer sets $N = pq$ and $m = p'q'$.
    2. The dealer chooses $d$ distinct primes $e_1, \ldots, e_d$ with each $e_i > n$ and computes $sk = \prod_{k=1}^{d} e_k^{-(\hat{\mathbf{v}}[k]+1)} \mod m$.
    3. Dealer also selects $t - 1$ random integers $a_1, \ldots, a_{t-1}$ with each $a_i$ randomly selected from $\{0, ..., m-1\}$ and computes a secret sharing polynomial $f(x) = sk + a_1 x + \ldots + a_{t-1} x^{t-1} \mod m$.
    4. Output is a public verification key $pk = (N, e_1, \ldots, e_d)$ and secret signing keys $sk_1, \ldots, sk_n$ for each signer $P_i$, where $sk_i = f(i) \mod m$.
Sign$(sk_i, (\mathbf{v}, c))$:
    1. Output the partial signature $\sigma_i = \mathcal{H}(c)^{n! sk_i \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \mod N$.
Stretch$((\sigma, \mathbf{v}, c), k, a)$:
    1. If $\mathbf{v}[k] + a > \hat{\mathbf{v}}[k]$, set $a = \hat{\mathbf{v}}[k] - \mathbf{v}[k]$.
    2. Output signature on stretched vector $\sigma' = \sigma^{e_k^a} \mod N$.
Combine$((\sigma_{i_1}, \mathbf{v}_{i_1}, c_{i_1}), \ldots, (\sigma_{i_t}, \mathbf{v}_{i_t}, c_{i_t}))$:
    1. Check that $c_{i_j} = c_{i_{j'}}$ for all $j, j' = 1, \ldots, t$.
    2. For each dimension $k$ of each partial signature $\sigma_{i_j}$:
        i. Run Stretch$((\sigma_{i_j}, \mathbf{v}_{i_j}, c_{i_j}), k, a)$ with $a = max(\mathbf{v}_{i_1}[k], \ldots, \mathbf{v}_{i_t}[k]) - \mathbf{v}_{i_j}[k]$.
    3. Compute modified Lagrange coefficients $(n!)\lambda_{i_j} = (n!)\frac{\prod_{j' \neq j}(0 - i_{j'})}{\prod_{j' \neq j}(i_j - i_{j'})}$ for each partial signature $\sigma_{i_j}$.
    4. Compute $w = \prod_{j=1}^{t} \sigma_{i_j}^{(n!)\lambda_{i_j}} = \mathcal{H}(c)^{(n!)^2 \prod_{k=1}^{d} e_k^{-(\hat{\mathbf{v}}[k] - \mathbf{v}[k] + 1)}} \mod N$.
    5. Since $\gcd((n!)^2, \prod_{k=1}^{d} e_k^{(\hat{\mathbf{v}}[k] - \mathbf{v}[k] + 1)}) = 1$, we use the extended Euclidean algorithm to compute $\alpha, \beta$ such that $\alpha (n!)^2 + \beta \prod_{k=1}^{d} e_k^{(\hat{\mathbf{v}}[k] - \mathbf{v}[k] + 1)} = 1$.
    6. Output the full signature $\sigma = w^\alpha \mathcal{H}(c)^\beta = \mathcal{H}(c)^{\prod_{k=1}^{d} e_k^{-(\hat{\mathbf{v}}[k] - \mathbf{v}[k] + 1)}} \mod N$.
Verify$(pk, \sigma, (\mathbf{v}', c'))$:
    1. Output 1 if $\mathcal{H}(c') = \sigma^{\prod_{k=1}^{d} e_k^{\hat{\mathbf{v}}[k] - \mathbf{v}'[k] + 1}} \mod N$. Otherwise, output 0.
```

Figure 1: Algorithms for our bounded vector signature construction.

The Sign and Verify algorithms require at most $\sum_{k=1}^{d} \hat{\mathbf{v}}[k]$ modular exponentiations in the group QR_N. Likewise, the Stretch algorithm requires a number of modular exponentiations that is linear in the range of the dimension being stretched. Finally, the Combine algorithm requires t modular exponentiations in QR_N to perform the Lagrange interpolation, and two additional exponentiations to derive the full signature using the coefficients from the extended Euclidean algorithm.

4. EXTENSIONS

The basic bounded vector signature construction given in the previous section may be extended in a variety of ways to achieve additional functionalities. These extensions include the ability to dynamically limit the use of the Stretch operation and to verify partial signatures via robustness techniques from the threshold signature literature. Here, we briefly describe each of these extensions in turn.

4.1 Dynamically Limiting Malleability

The malleability properties of the basic bounded vector signature construction allow anyone who possesses a partial or full signature to produce valid signatures on any vector with components greater than those in the originally signed vector, up to the maximum values specified during the KeyGen procedure. In certain instances, however, it is desirable to give each signer a degree of control over the use of the Stretch operation and, consequently, the signatures that other parties may produce. This can be accomplished by each signer taking steps to limit the degree to which its own partial signature can be stretched, and then leveraging the threshold t to limit what full signatures can then be produced using them.

Specifically, a signer i can create one bounded vector (partial) signature on its intended vector $\mathbf{v}_i$, and another bounded vector (partial) signature on the vector $\mathbf{v}'_i$, where $\hat{\mathbf{v}}[k] - \mathbf{v}'_i[k] \geq \mathbf{v}_i[k]$ for all dimensions k. This second signature can be viewed as bounding the extent to which the original vector can be stretched to only those vectors that can be simultaneously produced by both signatures. When $\mathbf{v}'_i[k] = \hat{\mathbf{v}}[k] - \mathbf{v}_i[k]$ for all dimensions k, all malleability has been removed from the partial signature.

Although the procedure above limits the malleability of partial signatures, its effect on the full signatures produced by Combine depends on the threshold t and its relation to the number of signers n. One particularly interesting parameterization allows us to ensure that a full signature (consisting of two bounded vector signatures) can be produced on only the

r^{th} smallest value in each dimension k (*i.e.,* value at rank r) from among the n values $\mathbf{v}_i[k]$, $i \in \{1 \ldots n\}$. To implement this, we set the threshold for the first bounded vector signature instantiation, to which signer i contributes a partial signature for $\mathbf{v}_i$, to $t_1 = r$. In the second bounded vector instantiation, each signer contributes a partial signature for $\mathbf{v}'_i$ where $\mathbf{v}'_i[k] = \hat{\mathbf{v}}[k] - \mathbf{v}_i[k]$, and we set the threshold to $t_2 = n - r + 1$. The first partial signatures can be stretched to increase the values $\mathbf{v}_i[k]$ of the $r-1$ smallest such values, while the second partial signatures can be stretched to decrease the values $\hat{\mathbf{v}}[k] - \mathbf{v}'_i[k]$ of the $n-r$ largest such values. The respective full signatures only overlap at the rank-r values in each dimension k, and consequently cannot be altered via the Stretch algorithm. When $r = 1$, this is a non-malleable signature on the component-wise minimums, and when $r = n$, this is a non-malleable signature on the component-wise maximums, for example. In Section 5, we leverage this capability to perform intersection and union of signed data structures.

4.2 Adding Robustness

The notion of *robustness* in threshold signature schemes captures the inability of an adversary to produce incorrectly formatted partial signatures, which may cause the Combine algorithm to fail. Since the core of our bounded vector signature construction is based on the threshold RSA scheme of Shoup [38], we can make use of the robustness techniques presented therein. Broadly speaking, the robustness property can be achieved by requiring the signer to prove, in zero knowledge, that (i) the key used to sign the partial signature is the same as the one provided by the dealer, and (ii) that the partial signature is properly formed according to the algorithm description.

Specifically, the dealer chooses a random element $g \in QR_N$. Note that with overwhelming probability g is a generator of QR_N. He then computes and publishes the value $G_i = g^{sk_i}$ for each signer P_i. The partial signature σ_i is then accompanied by a zero-knowledge proof that $\log_{\mathcal{H}(c)}(\sigma_i^2) = \log_g(G_i^{2(n!)^2 \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}})$, which can be accomplished using standard zero-knowledge techniques for proving the equality of discrete logarithms [12, 11]. Furthermore, this proof can be made non-interactive by using the Fiat-Shamir heuristic [20]. Due to space considerations and the similarity of our techniques with those of Shoup, we forgo a proof of the simulatability of this zero-knowledge proof.

5. APPLICATIONS

Bounded vector signatures can be used to efficiently represent signatures on a variety of data structures while still allowing certain operations on those data structures. These efficient, malleable signatures are particularly important in distributed systems applications, where disparate parties must securely share information while limiting communications overhead. Here, we describe two such data structures, namely sets and intervals. For each data structure, we describe how to represent it using our bounded vector signatures, specify operations on those signed data structures enabled by the Stretch operation, and suggest applications to problems in the area of distributed systems security.

The use of bounded vector signatures in the distributed applications described below provides a number of benefits. These include the compact representation of large groups of items (*e.g.,* sets and intervals) with a constant number of group elements in QR_N, the ability to represent aggregated information from all members of a group as a single signature, non-interaction of the Combine and Stretch protocols, and the ability to locally update the full signature on the aggregated information. Furthermore, the threshold used in the Combine algorithm acts as a voting mechanism for the aggregated information, and may be of use in scenarios where Byzantine fault tolerance is required. Unless otherwise noted in this section, the bounded vector signatures are parameterized with $t = n$.

5.1 Sets and Multisets

The set data structure allows for the storage of unordered collections of unique items or values. In a multiset, this concept is extended to allow multiple copies of items (*i.e.,* associating a count with each item). Sets (resp., multisets) are represented in our bounded vector signature scheme by associating each item in the universe of possible items with a dimension in the signed vector and limiting the maximum value of the dimension to one (resp., the maximum per-item count). In essence, the vector representing a set reflects the presence or absence of each of the d items in the universe, while a vector for a multiset dictates the count of each item.

One downside of this approach for signing sets and multisets is that it requires a number of dimensions equal to the size of the universe of items represented, which may be quite large. In those cases, we may instead use our bounded vector signatures to encode the set in a *Bloom filter* [7]. Rather than associating each component of the vector with a specific item, we instead use the outputs of hash functions applied to the item to determine which dimensions to set to one. To test membership, we check that all dimensions to which the item hashes are set to one. In short, Bloom filters allow for a compact representation of sets by trading off a tunable probability of false positives when testing set membership (*i.e.,* an item is reported as a member even though it is not). Like simple sets, our bounded vector signatures represent a Bloom filter using a d-dimensional vector with maximum values for each dimension set to one, only in this case the number of dimensions may be much smaller than the number of items in the universe and is related to the desired probability of false positives.

Operations on Signed Sets.

The stretch property of the underlying bounded vector signatures allows us to perform the union operation among multiple set signatures simply by performing the Combine operation. As discussed in the previous section, Combine effectively takes the maximum value in each dimension from among the t partial signatures to form a final signature. As it turns out, this exactly defines the union among the signed sets. Furthermore, the security of our bounded vector signature scheme ensures that while items may be added to the (partially or fully) signed set, it is computationally infeasible for an adversary to remove an item. If additions to the fully signed set should not be permitted, then the rank-order technique described in Section 4.1 can be used to preclude this. The union operation and security guarantee also hold for multisets and Bloom filters, as well.

The intersection operation is achieved by having each signer produce a signature on a vector where each component encodes the distance between the signed value and the maximum allowable value in that dimension (*i.e.,* the complement for sets or the number of copies of items not in the multiset). As mentioned in Section 4.1, the Stretch operation on these signatures *decreases* the value being signed by increasing the distance from the maximum. Therefore, when Combine is run on these signatures, the largest value that can be signed is, in effect, the minimum value in each dimension. The security of bounded vector signatures ensures that the resultant signatures can have items removed from the set (resp., multiset, Bloom filter), but not added. Again, the technique of Section 4.1 can be used to preclude removals from the fully signed intersection, if so desired.

Distributed Systems Applications.

Signatures on sets, multisets, and Bloom filters can be used in a number of distributed data sharing applications, including peer-to-peer systems [14, 30, 24, 36], distributed caches [19, 9, 35], blacklisting services [17, 40, 39], and network flow monitoring systems [18]. In each of these applications, a collection of entities must provide users with an aggregate view of the data that they maintain as a group. These systems cannot assume interactivity, must provide an efficient method of verification to the user, and must be able to perform updates on the aggregated data without requiring additional interaction.

As a concrete example, consider the problem of spam blacklisting services, such as Spamhaus [40], SORBS [39], and DNSBL [17], which aggregate information about IP addresses sending spam e-mail. These blacklists must aggregate the IP addresses from several sources and provide users with a method of verifying that the given list does indeed represent that aggregate information (*i.e.,* names have not been spuriously added or removed, as the case may be). The blacklists can be efficiently represented in a Bloom filter, signed by our bounded vector signature method, and combined in a completely offline manner by the blacklist providers to give users a single signature to verify the aggregated blacklist. In the time between blacklist aggregation events, the individual sources can continue adding or removing IP addresses from the aggregated blacklist through the use of the Stretch operation without requiring additional interactions, if such operations are allowed by the signature instantiation. By contrast, traditional digital signatures would force the user to verify the blacklists from each of the sources independently and require the blacklist sources to continually generate new signatures on the published blacklists.

5.2 Intervals

Bounded vector signatures can also be used to represent contiguous intervals of values, in addition to discrete sets of items. There are potentially two ways in which to encode an interval of values within a bounded vector signature scheme. In the first, the signer creates 2-dimensional vectors, where the components of the vector encode the left and right endpoints of the signed interval, respectively. In the second, the signer may create two 1-dimensional vectors with each vector encoding one of the endpoints. The primary tradeoff between these two methods is that in the first case the range is efficiently encoded in a single group element, whereas the second case allows for two unique parameterizations of the bounded vector signatures for the vectors (*e.g.,* two different thresholds t). As we will show later in this section, there are potential applications that are specific to both approaches. The concept of signing an interval can also be expanded to an arbitrary number of dimensions by simply increasing the dimensionality of the signed vector, or number of signatures in the case where each endpoint is a single vector. Specifically, if we want to encode a d-dimensional box (*i.e.,* the Cartesian product of d intervals), we create a $2d$-dimensional vector to encode the left and right endpoints for the allowable range in each dimension.

Operations on Signed Intervals.

The use of bounded vector signatures provides the signer with a number of options for encoding the endpoint values of the signed interval, which in turn enable different functionalities. For example, if the signer were to encode the left-most point in the interval as the value itself and the right-most point as the distance to the maximum allowable value in the bounded vector scheme (*i.e.,* the vector $\langle \mathbf{v}[1], \hat{\mathbf{v}}[2] - \mathbf{v}[2] \rangle$ with the restriction $\mathbf{v}[1] \le \hat{\mathbf{v}}[2] - \mathbf{v}[2]$), then the resultant signed range could only be *shrunk* and never expanded — this is equivalent to the dynamic malleability limitation described in Section 4.1. On the other hand, if the signer encoded the left-most point as the distance from the maximum and the right-most as the value itself (*i.e.,* $\langle \hat{\mathbf{v}}[1] - \mathbf{v}[1], \mathbf{v}[2] \rangle$ with $\hat{\mathbf{v}}[1] - \mathbf{v}[1] \le \mathbf{v}[2]$), then the signed range could be *expanded* but never shrunk. Notice that if we set both endpoints to be encoded relative to the same value (*e.g.,* $\langle \mathbf{v}[1], \mathbf{v}[2] \rangle$ or $\langle \hat{\mathbf{v}}[1] - \mathbf{v}[1], \hat{\mathbf{v}}[2] - \mathbf{v}[2] \rangle$), we can slide and expand the interval in one direction only.

The range encodings above, along with the Stretch operation on the underlying bounded vector signatures, allow a combiner to efficiently compute the intersection and union of signed intervals. To compute intersections, each signer encodes the endpoints of its signed interval such that the range can only be shrunk. When the Combine algorithm is run on these partial signatures, the combiner will be forced to shrink the intervals to equal the smallest overlapping range and so the output will be a full signature on the intersection. For the union operation, the signers use an encoding that allows expansion of the signed interval and the combiner will expand all intervals such that the full signature will contain the left and right-most points from among all partially signed intervals. We note that this is slightly different from the typical definition of union among intervals, since in our system the union operation is allowed even if the intervals do not overlap. The security of the bounded vector signatures ensures that the full signature output by the combine procedure cannot be altered to sign a value not found within the intersection or union of the individual partially signed intervals.

Distributed Systems Applications.

Like signatures on sets and multisets, our bounded vector signature representations of intervals can be used to accomplish data sharing tasks within distributed systems for values that have natural ordering properties. One example is a time interval. For example, an approach to the management of public-key certificates (*e.g.,* [29]) employs an *online revocation authority ORA* to *countersign*, for short intervals, long-term certificates created by a more trusted offline certificate authority CA. In a traditional implementation of this idea, both the signature by CA and the countersignature by ORA would be implemented using separate signatures, resulting in two signature verifications per use of the certificate. Our bounded vector signatures support the consolidation of these two signatures into one: the CA provides to ORA a signed certificate with a large validity period, which ORA can parcel out in small portions by "sliding" the ends of the validity interval accordingly (rather than separately countersigning each short validity interval).

A second area for using bounded vector signatures to represent intervals is sensor applications. Marzullo [31], for example, considered a setting in which multiple sensors produce an interval in which it senses a value (e.g., temperature, time) to lie, and where a sensor is *correct* if the interval it returns contains the actual value (and is *faulty* otherwise). His algorithm uses intervals from n sensors to produce the tightest interval possible in which the actual value lies, provided that fewer than f sensors are faulty. A direct use of our bounded vector signatures would permit each sensor to sign its interval and subsequently combine these signed intervals using this fault-tolerant intersection algorithm to produce a signature on the tightest interval possible in which the actual value lies. Others can be disallowed from tightening the interval further using the rank-order techniques of Section 4.1.

A third interesting application of signatures on intervals is their use to represent a broad class of access-control structures used in key management tasks, like hierarchical access controls [2, 4] and group key management [34, 41]. In these settings, attributes that determine access can often be encoded within a d-dimensional space. For instance, in some location-based access control schemes [3, 42], the dimensions encode the location of the entity in 3-dimensional space, plus an extra dimension for time. A service provider can sign subranges describing the allowable access attributes for the entities in the distributed system and provide them with their respective subranges as a non-interactive access token. These entities can then derive a "key" [2] for any subspace of their allowable attributes using the Stretch operation on the signature given to them. Such a scheme can derive several benefits from our bounded vector signatures, such as security against key recovery [2], which roughly states that it is infeasible for the adversary to derive a key for an access attribute for which she does not have a token.

6. CONCLUSION

In this paper, we described a new malleable signature scheme for signing vectors of natural numbers, which we call bounded vector signatures. The primary contribution of our scheme is a malleability property that allows arbitrary parties to *increase* the value embedded in any component of the signed vector without access to the signing key, while making it computationally infeasible to *decrease* values. In the multiparty setting, this malleability property allows each signer to sign a potentially different vector of values and yet still produce a valid full signature representing the component-wise maximum of any t vectors. We described an efficient construction for our scheme and proved it secure under the strong RSA and decisional Diffie-Hellman assumptions in the random oracle model. Finally, we showed that bounded vector signatures could be used to sign interesting data structures, such as sets and intervals, and that the malleability of those signatures enabled efficient implementations of standard operations on the signed structures. Moreover, we illustrated the unique benefits of these malleable signatures in providing compact and non-interactive solutions to several problems in the area of distributed systems security.

[2]Depending on the scheme in use, additional key derivation steps may be necessary. For simplicity we just denote the signature as a "key".

Acknowledgements

This work was supported in part by NSF grants 0756998, 0910483, and by a grant from Intel. Support was also provided by NSF grant 0937060 that was awarded to the Computing Research Association for the CIFellows Project.

7. REFERENCES

[1] S. Agrawal, D. Boneh, X. Boyen, and D. Mandell-Freeman. Preventing pollution attacks in multi-source network coding. Cryptology ePrint Archive, Report 2010/183, 2010. `http://eprint.iacr.org/`.

[2] M.J. Atallah, M. Blanton, N. Fazio, and K.B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Transactions on Information and System Security*, 12(3):1–43, 2009.

[3] M.J. Atallah, M. Blanton, and K.B. Frikken. Efficient techniques for realizing geo-spatial access control. In 2nd *ACM Symposium on Information, Computer and Communications Security*, pages 82–92, 2007.

[4] G. Ateniese, A. De Santis, A.L. Ferrara, and B. Masucci. Provably-secure time-bound hierarchical key assignment schemes. In 13th *ACM Conference on Computer and Communications Security*, pages 288–297, 2006.

[5] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology—Eurocrypt '97*, pages 480–494, 1997.

[6] M. Bellare and Neven G. Transitive signatures: new schemes and proofs. *IEEE Transactions on Information Theory*, 51(6):2133–2151, 2005.

[7] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[8] D. Boneh, D. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In 12th *International Conference on Practice and Theory in Public Key Cryptography*, pages 68–87, 2009.

[9] J.W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. *IEEE/ACM Transactions on Networking*, 12(5):767–780, 2004.

[10] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 101–120, 2002.

[11] J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In *Advances in Cryptology—Crypto '99*, volume 1666, pages 413–430, 1999.

[12] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology—Crypto '92*,

pages 89–105, 1992.

[13] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):161–185, 2000.

[14] F.M. Cuenca-Acuna, C. Peery, R.P. Martin, and T.D. Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In 12th *IEEE Symposium on High Performance Distributed Computing*, pages 236–246, 2003.

[15] I. Damgård and M. Koprowski. Practical threshold RSA signatures without a trusted dealer. In *Advances in Cryptology—Eurocrypt '01*, pages 152–165, 2001.

[16] W. de Jonge and D. Chaum. Some variations on RSA signatures and their security. In *Advances in Cryptology—Crypto '86*, pages 49–59, 1986.

[17] DNS-based blacklist, 2010. `http://www.dnsbl.info`.

[18] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems*, 21(3):270–313, 2003.

[19] L. Fan, P. Cao, J. Almeida, and A.Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[20] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—Crypto '86*, volume 263 of LNCS, pages 186–194, 1986.

[21] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology—Eurocrypt '99*, pages 123–139, 1999.

[22] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. Secure network coding over the integers. Cryptology ePrint Archive, Report 2009/569, 2010. `http://eprint.iacr.org/`.

[23] A. Hevia and D. Micciancio. The provable security of graph-based one-time signatures and extensions to algebraic signature schemes. In *Advances in Cryptology—Asiacrypt '02*, pages 379–396, 2002.

[24] T.D. Hodes, S.E. Czerwinski, B.Y. Zhao, A.D. Joseph, and R.H. Katz. An architecture for secure wide-area service discovery. *Wireless Networks*, 8(2/3):213–230, 2002.

[25] S. Hohenberger and B. Waters. Short and stateless signatures from the RSA assumption. In *Advances in Cryptology—Crypto '09*, pages 654–670, 2009.

[26] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *The Cryptographer's Track at the RSA Conference on Topics in Cryptology*, pages 244–262, 2002.

[27] A. Kiayias and M. Yung. Group signatures: provable security, efficient constructions and anonymity from trapdoor-holders. Cryptology ePrint Archive, Report 2004/076, 2004. `http://eprint.iacr.org/`.

[28] E. Kiltz, A. Mityagin, S. Panjwani, and B. Raghavan. Append-only signatures. In 32th *International Colloquium on Automata, Languages and Programming*, 2005.

[29] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.

[30] J. Ledlie, J.M. Taylor, L. Serban, and M. Seltzer. Self-organization in peer-to-peer systems. In 10th *ACM SIGOPS Workshop*, pages 125–132, 2002.

[31] K. Marzullo. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems*, 8(4):284–304, 1990.

[32] S. Micali and R. L. Rivest. Transitive signature schemes. In *The Cryptographer's Track at the RSA Conference on Topics in Cryptology*, pages 236–243, 2002.

[33] G. Neven. A simple transitive signature scheme for directed trees. *Theoretical Computer Science*, 396(1-3):277 – 282, 2008.

[34] A. Perrig, D. Song, and D. Tygar. ELK: a new protocol for efficient large-group key distribution. In *2001 IEEE Symposium on Security and Privacy*, pages 247–262, 2002.

[35] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *2003 ACM/IFIP/USENIX International Conference on Middleware*, pages 21–40, 2003.

[36] S.C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In 21st *Joint Conference of the IEEE Computer and Communication Societies*, pages 1248–1257, 2002.

[37] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[38] V. Shoup. Practical threshold signatures. In *Advances in Cryptology—Eurocrypt '00*, pages 207–220, 2000.

[39] Spam and open relay blocking system, 2010. `http://www.sorbs.net`.

[40] The spamhaus project, 2010. `http://www.spamhaus.org`.

[41] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, 2000.

[42] H. Yuan and M.J. Atallah. Efficient and secure distribution of massive geo-spatial data. In 17th *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 440–443, 2009.

APPENDIX

A. PROOF OF THEOREM

In this appendix, we provide the proof for Theorem 1 from Section 3.1.

Proof. Given an adversary $\mathcal{A}$ capable of breaking shrink unforgeability in time T, we construct an adversary $\mathcal{B}$ for the Strong RSA problem. $\mathcal{A}$ is provided with a hash oracle modeled as a random oracle and n signing oracles, and is capable of breaking shrink unforgeability with advantage $\mathbf{Adv}_{\mathcal{BVS}}(\mathcal{A})$. $\mathcal{B}$ is given a RSA modulus N and $y \in \mathbb{Z}_N^*$ as input. His goal is to find a $x \in \mathbb{Z}_N^*$ and $e > 1$, such that $x^e = y \bmod N$.

Initialization:

1. $\mathcal{B}$ chooses d distinct primes $e_1, \ldots, e_d$ such that $e_i > n$, $i = 1, \ldots, d$. He then initializes $\mathcal{A}$ with $pk = (N, e_1,$

…, e_d).

2. $\mathcal{B}$ randomly selects $t-1$ random values $(sk_{i_1}, \ldots, sk_{i_{t-1}}) \xleftarrow{\$} \{0, \ldots, \lfloor N/4 \rfloor - 1\}$. These values act as the signing keys known to $\mathcal{B}$ (*i.e.,* referred to as corrupted signers in the threshold literature). Note that the values are supposed to be chosen from the subgroup $\mathbb{Z}_m$, however, the statistical difference between these two uniform distributions is $O(N^{-1/2})$ and they are therefore statistically indistinguishable. We use KEYED to denote the set of signing oracles for which $\mathcal{B}$ knows the secret signing key and use KEYLESS to denote the rest of them, for which $\mathcal{B}$ does not know the signing key.

3. $\mathcal{B}$ also chooses a random value $h^* \xleftarrow{\$} \{1, \ldots, q_h\}$ that acts as her guess for the hash query containing the context that $\mathcal{A}$ will produce a forgery on. From now on, we refer to this h^{*th} context as c^*.

4. Finally, $\mathcal{B}$ chooses a random value $b \xleftarrow{\$} \{0,1\}$ and guesses which type of forger he is facing. If $b = 0$, then he guesses that $\mathcal{A}$ is a Type 1 forger that will only query up to $t-1$ signing oracles on vectors with the context for which he forges. Otherwise, $\mathcal{B}$ guess that $\mathcal{A}$ is a Type 2 forger that will query at least t signing oracles on the context for which he forges.

At a high-level, $\mathcal{B}$'s strategy against a forger $\mathcal{A}$ is to guess on which hash query $\mathcal{A}$ is going to make the forgery on. Remember that a signature on a vector context pair $(\mathbf{v}, c)$ is essentially a root of the hash of the context, with the components of the vector determining the depth of the root. $\mathcal{B}$ will embed his Strong RSA instance into the hash value so that a forgery by $\mathcal{A}$ will be helpful to him to solve his problem.

Notice that at the end of the initialization phase, $\mathcal{B}$ guesses which type of forger he is facing. The reason that we consider these two types separately is that the shrink unforgeability definition treats these two situations differently. Indeed, our construction prohibits an adversary who obtains partial signatures from less than t signing oracles to generate a full signature on *any* vector-context pair, regardless of $\mathcal{A}$'s queries. In this case, a forgery on any vector would be considered a valid forgery. On the other hand, if the forger obtains partial signatures from t or more signing oracles on c^*, then he is legitimately capable of producing full signatures on pairs $(\mathbf{v}, c^*)$ with each component of $\mathbf{v}$ equal or greater than the t^{th} smallest among each component of the vectors he queried on c^*. In this case, only a forgery on a vector in which at least one component is less than the t^{th} smallest queried in that dimension is considered a valid forgery by $\mathcal{A}$.

A.1 Interacting with Type 1 Forger

The basic strategy of $\mathcal{B}$ is to embed the Strong RSA instance into the hash value of c^* in a way that will enable him to solve his Strong RSA problem if $\mathcal{A}$ does output a forgery on a pair $(\mathbf{v}, c^*)$. The hash and signing oracles operate as follows in this scenario.

Hash Oracle $\mathcal{H}(c)$ (Type 1) :

1. If $c = c^*$: $\mathcal{B}$ returns $\mathcal{H}(c) = y$ and records $(c, \mathcal{H}(c))$ to answer future queries.
2. If $c \neq c^*$:
   (a) If c was previously queried, return $\mathcal{H}(c)$.
   (b) Otherwise, $\mathcal{B}$ selects a random $r \xleftarrow{\$} \mathbb{Z}_N^*$, returns $\mathcal{H}(c) = r^{2 \cdot \prod_{k=1}^{d} e_k^{\hat{v}+1}} \bmod N$, and records $(c, r, \mathcal{H}(c))$. The extra squaring on the exponent is to make sure the hash value is in the QR_N group.

Signing Oracle$_i(\mathbf{v}, c)$ (Type 1) :

1. If $c = c^*$:
   (a) Regardless of whether this oracle is KEYLESS or KEYED, if no signature query on c^* has been made to this oracle, $\mathcal{B}$ picks a new secret signing key $sk_i' \xleftarrow{\$} \{0, \ldots, \lfloor N/4 \rfloor - 1\}$ and returns $\sigma_i = \mathcal{H}(c)^{(n!) sk_i' \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$. It stores the newly generated secret key for signing future queries $(\mathbf{v}, c^*)$.
   (b) If there has been a signature query on c^* made to this oracle, $\mathcal{B}$ uses the previously selected new secret signing key sk_i' to return $\sigma_i = \mathcal{H}(c)^{(n!) sk_i' \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$.
2. If $c \neq c^*$:
   (a) If this oracle is KEYED, $\mathcal{B}$ returns $\sigma_i = \mathcal{H}(c)^{(n!) sk_i \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$.
   (b) If this oracle is KEYLESS, $\mathcal{B}$ makes use of a well formed full signature and $t-1$ secret signing keys to interpolate the proper signature share to return by using the technique described in [38] :
      i. $\mathcal{B}$ finds the entry (c, r) in the hash table. He computes the intended full signature $\sigma = r^{2 \cdot \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$ on $(\mathbf{v}, c)$. It is easy to verify the validity of this signature.
      ii. $\mathcal{B}$ then computes modified Lagrange coefficients $(n!)\lambda_{i_j} = (n!) \frac{\prod_{j' \neq j}(i - i_{j'})}{\prod_{j' \neq j}(i_j - i_{j'})}$ for each signing key sk_{i_j}, $j = 1, \ldots, t-1$, to which he has access and $(n!)\lambda_0 = (n!) \frac{\prod_{j'}(i - i_{j'})}{\prod_{j'}(0 - i_{j'})}$.
      iii. Finally, $\mathcal{B}$ returns $\sigma_i = \sigma^{(n!)\lambda_0} \cdot \prod_{j=1}^{t-1} \mathcal{H}(c)^{(n!)\lambda_{i_j} sk_{i_j} \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$ as the partial signature for $(\mathbf{v}, c)$.

The major difficulty in dealing with a Type 1 forger is that $\mathcal{B}$ does not know in advance which signing oracles $\mathcal{A}$ is going to query for the context c^*. If $\mathcal{A}$ asks for a signature on a pair $(\mathbf{v}, c^*)$ to a KEYLESS oracle, then $\mathcal{B}$ will not be able to answer it using the keys $sk_{i_1}, \ldots, sk_{i_{t-1}}$. Our strategy is to let $\mathcal{B}$ dynamically select a new set of $t-1$ random secret signing keys for the oracles that $\mathcal{A}$ queries for the guessed context c^*. $\mathcal{B}$ can then use the $t-1$ new signing keys to answer partial signatures for the oracle even if it was a KEYLESS oracle. Although this strategy seems to have solved the problem, there is a caveat. Since we changed the signing key of some oracles when answering signature requests on c^*, it could happen that one KEYED oracle used its original signing key to return signature queries on other contexts, but used the new signing key to sign on c^*. Thus, the distribution of the simulation will differ in this respect from the real protocol. However, as we will show in Lemma 1 from Section A.3, the distributions are indistinguishable under a variant of the DDH assumption in the group QR_N.

In the end, $\mathcal{A}$ outputs a forgery σ' on the pair $(\mathbf{v}', c')$. Suppose $\mathcal{B}$ made the correct guess on context c' (*i.e.,* $c' =$

c^*). If the forgery is a valid signature, then we should have $\sigma'^{\prod_{k=1}^{d} e_k^{\hat{v}-\mathbf{v}'[k]+1}} = \mathcal{H}(c') = y$. It is easy to see that $\mathcal{B}$ has solved his Strong RSA problem by choosing x to be σ' and e to be $\prod_{k=1}^{d} e_k^{\hat{v}-\mathbf{v}'[k]+1}$. Now, we bound the probability that $\mathcal{B}$ is able to solve the Strong RSA problem when facing a Type 1 forger. We use $\mathcal{A}^{Type1}$ to denote the event that $\mathcal{A}$ is a Type 1 forger, and E_1 to represent the event that a Type 1 forger successfully outputs a forgery under our simulated view.

$$\begin{aligned} & \mathbf{Pr}\left[\mathcal{B}\ wins \mid \mathcal{A}^{Type1}\right] \\ = \ & \mathbf{Pr}\left[\mathcal{B}\ wins \mid \mathcal{A}^{Type1} \wedge \mathsf{E}_1\right] \cdot \mathbf{Pr}\left[\mathsf{E}_1\right] \\ = \ & \frac{1}{q_h} \cdot \mathbf{Pr}\left[\mathsf{E}_1\right] \end{aligned} \quad (1)$$

In our simulation, $\mathbf{Pr}\left[\mathcal{B}\ wins \mid \mathcal{A}^{Type1} \wedge \mathsf{E}_1\right]$ is contingent upon $\mathcal{B}$'s ability to guess the context within which to embed the Strong RSA problem instance, which occurs with probability at least $\frac{1}{q_h}$, where q_h is the total number of hash queries made during the simulation. $\mathbf{Pr}\left[\mathsf{E}_1\right]$ is affected by the advantage of $\mathcal{A}^{Type1}$ in distinguishing the simulated and real distributions from Lemma 1. We show in Section A.3 that

$$\mathbf{Pr}\left[\mathsf{E}_1\right] \geq \mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A}^{Type1}) - (t-1) \cdot \mathbf{Adv}_{\mathsf{DDH}}(\mathcal{D}) \quad (2)$$

where $\mathcal{D}$ is a DDH adversary taking the same time (asymptotically) as $\mathcal{B}$. By substituting this into (1), we get the bound of the probability that $\mathcal{B}$ solves the Strong RSA instance with a Type 1 forger:

$$\begin{aligned} & \mathbf{Pr}\left[\mathcal{B}\ wins \mid \mathcal{A}^{Type1}\right] \\ \geq \ & \frac{1}{q_h} \cdot \left(\mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A}^{Type1}) - (t-1) \cdot \mathbf{Adv}_{\mathsf{DDH}}(\mathcal{D})\right) \end{aligned} \quad (3)$$

A.2 Interacting with Type 2 Forger

In this scenario, $\mathcal{B}$ still makes a guess at the hash query h^* containing the context c^* that forger $\mathcal{A}$ is going to forge on as before. However, $\mathcal{B}$ also guesses a dimension $k^* \xleftarrow{\$} \{1, \ldots, d\}$ within which the adversary $\mathcal{A}$ will violate shrink unforgeability, and a value $v^* \xleftarrow{\$} \{1, \ldots, \hat{v}\}$ which the forgery in that dimension will be less than. $\mathcal{B}$ will embed his Strong RSA instance into the hash value of c^* in a way that limits his ability to answer signature queries on values less than v^* in dimension k^* on c^*, but ensures that he would be able to make use of forgeries in that dimension to solve his Strong RSA problem. No such limitation exists for other components of the signed vectors. The hash and signing oracles operate as follows.

Hash Oracle $\mathcal{H}(c)$ (Type 2) :

1. If $c = c^*$:
   (a) $\mathcal{B}$ embeds the strong RSA instance into the response. He returns $\mathcal{H}(c) = y^{e_{k^*}^{\hat{v}-v^*+1} \prod_{k \neq k^*} e_k^{\hat{v}+1}}$ mod N and records $(c, \mathcal{H}(c))$ to answer future queries. Note that by setting the hash value c in this way, whenever $\mathcal{A}$ makes a signing query on a pair $(\mathbf{v}, c')$ such that $c' = c$, $\mathcal{B}$ is able to answer it only if $\mathbf{v}[k^*] \geq v^*$.
2. If $c \neq c^*$:
   (a) If c was previously queried, $\mathcal{B}$ returns $\mathcal{H}(c)$.
   (b) Otherwise, $\mathcal{B}$ selects a random $r \xleftarrow{\$} \mathbb{Z}_N^*$, returns $\mathcal{H}(c) = r^{2 \cdot \prod_{k=1}^{d} e_k^{\hat{v}+1}}$, and records $(c, r, \mathcal{H}(c))$.

Signing Oracle$_i(\mathbf{v}, c)$ (Type 2) :

1. If this oracle is KEYED, $\mathcal{B}$ returns $\sigma_i = \mathcal{H}(c)^{(n!)sk_i \prod_{k=1}^{d} e_k^{\mathbf{v}[k]}}$ mod N.
2. If this oracle is $\mathsf{KEYLESS}$ and $\mathbf{v}[k^*] < v^*$, then $\mathcal{B}$ aborts since it is unable to provide the appropriate signature.
3. If this oracle is $\mathsf{KEYLESS}$ and $\mathbf{v}[k^*] \geq v^*$, $\mathcal{B}$ makes use of a well formed full signature and $t-1$ secret signing keys to interpolate the proper signature share to return. It uses exactly the same interpolation technique described in the signing oracle specification of a $\mathsf{KEYLESS}$ oracle for Type 1 forger.

Assuming that $\mathcal{B}$ does not abort, $\mathcal{A}$ will output a forgery σ' on the vector-context pair $(\mathbf{v}', c')$ such that at least one component in $\mathbf{v}'$ is less than $v_t^{c'}[k]$, which was defined in Definition 1 from Section 2. The adversary $\mathcal{B}$ checks that $\mathsf{Verify}(pk, \sigma', (\mathbf{v}', c')) = 1$, that $c' = c^*$, and that $\mathbf{v}'[k^*] < v^*$. In other words, $\mathcal{B}$ has made the correct guesses on the context c^* and the dimension k^* of the forgery, and the output value on dimension k^* is less than v^*. If any of these checks fail, $\mathcal{B}$ aborts.

If $\mathcal{A}$ has produced a valid forgery, then the verification condition should hold:

$$\sigma'^{\prod_{k=1}^{d} e_k^{\hat{v}-\mathbf{v}'[k]+1}} = \mathcal{H}(c') = y^{e_{k^*}^{\hat{v}-v^*+1} \prod_{k \neq k^*} e_k^{\hat{v}+1}}$$

Rearranging both sides gives us:

$$\sigma'^{e_{k^*}^{v^*-\mathbf{v}'[k^*]}} = y^{\prod_{k \neq k^*} e_k^{\mathbf{v}'[k]}}$$

Since $\gcd(e_{k^*}^{v^*-\mathbf{v}'[k^*]}, \prod_{k \neq k^*} e_k^{\mathbf{v}'[k]}) = 1$, $\mathcal{B}$ can then use the extended Euclidean algorithm to compute α, β such that $\alpha e_{k^*}^{v^*-\mathbf{v}'[k^*]} + \beta \prod_{k \neq k^*} e_k^{\mathbf{v}'[k]} = 1$. The Strong RSA problem is then solved by setting $x = y^{\alpha} \cdot \sigma'^{\beta}$ and $e = e_{k^*}^{v^*-\mathbf{v}'[k^*]}$.

The probability of our constructed Strong RSA adversary $\mathcal{B}$ producing the correct solution to the given Strong RSA instance is clearly based on the odds that $\mathcal{B}$ did not abort while answering oracle queries and $\mathcal{A}$ output a forgery that was helpful to $\mathcal{B}$. To bound this probability, we note that $\mathcal{B}$ has to have made the correct guess on the context c^* and the dimension k^*, which we denote as the event E_2. Next, we define E_3 as the event that v^* is less than or equal to any value queried to a $\mathsf{KEYLESS}$ oracle on dimension k^* (*i.e.,* $\mathcal{B}$ did not abort during oracle queries). We also define event E_4, which represents the case that the forgery is helpful for $\mathcal{B}$ in breaking his Strong RSA problem instance. The probability that $\mathcal{B}$ did not abort during the simulation is:

$$\begin{aligned} \mathbf{Pr}\left[\mathcal{B}\ \neg abort\right] &= \mathbf{Pr}\left[\mathsf{E}_2 \wedge \mathsf{E}_3 \wedge \mathsf{E}_4\right] \\ &= \mathbf{Pr}\left[\mathsf{E}_2\right] \cdot \mathbf{Pr}\left[\mathsf{E}_3 \wedge \mathsf{E}_4\right] \\ &= \frac{1}{q_h \cdot d} \cdot \mathbf{Pr}\left[\mathsf{E}_3 \wedge \mathsf{E}_4\right] \\ &= \frac{1}{q_h \cdot d} \cdot \mathbf{Pr}\left[\mathsf{E}_4 \mid \mathsf{E}_3\right] \cdot \mathbf{Pr}\left[\mathsf{E}_3\right] \end{aligned} \quad (4)$$

$\mathbf{Pr}[\,\mathsf{E}_3\,]$ can be bounded as follows.

$$\begin{aligned}\mathbf{Pr}[\,\mathsf{E}_3\,] &= \sum_{i=1}^{\hat{v}} \mathbf{Pr}[\,v^* = i\,] \cdot \mathbf{Pr}[\,\mathsf{E}_3 \mid v^* = i\,] \\ &\geq \mathbf{Pr}[\,v^* = 1\,] \cdot \mathbf{Pr}[\,\mathsf{E}_3 \mid v^* = 1\,] \\ &\geq \frac{1}{\hat{v}} \cdot \left(\frac{t-1}{n}\right)^{t-1} \qquad (5)\end{aligned}$$

Here, we focus on the case where $v^* = 1$ for ease of analysis. Since v^* is selected uniformly from the set $\{1, \ldots, \hat{v}\}$, the probability that 1 is selected is $\frac{1}{\hat{v}}$. When this happens, the $\mathsf{KEYLESS}$ signing oracles are able to answer signature queries only when $\mathbf{v}[k^*] \geq 1$. Therefore, the worst non-abort scenario is one in which $\mathcal{A}$ queries for $t-1$ distinct signatures on the value 0 for context c^*, which requires that those queries be answered by the KEYED oracles. This event occurs with probability at least $(\frac{t-1}{n})^{t-1}$.

Likewise, we compute $\mathbf{Pr}[\,\mathsf{E}_4 \mid \mathsf{E}_3\,]$ using the fact that, in the worst case, any forgery produced by a non-aborted simulation must, by definition, involve queries to $t-1$ KEYED signing oracles, which has a probability of:

$$\mathbf{Pr}[\,\mathsf{E}_4 \mid \mathsf{E}_3\,] \geq \left(\frac{t-1}{n}\right)^{t-1} \qquad (6)$$

By substituting into (4), we get the probability that $\mathcal{B}$ does not need to abort by the end of the execution:

$$\mathbf{Pr}[\,\mathcal{B}\ \neg abort\,] \geq \frac{1}{q_h \cdot d} \cdot \frac{1}{\hat{v}} \cdot \left(\frac{t-1}{n}\right)^{2(t-1)} \qquad (7)$$

Consequently, we get the probability that $\mathcal{B}$ succeeds in solving his Strong RSA problem when interacting with a Type 2 forger:

$$\begin{aligned}&\mathbf{Pr}[\,\mathcal{B}\ wins \mid \mathcal{A}\ is\ type2\,] \\ =\ &\mathbf{Pr}[\,\mathcal{B}\ \neg abort\,] \cdot \mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A}^{Type2}) \\ \geq\ &\frac{1}{q_h \cdot d} \cdot \frac{1}{\hat{v}} \cdot \left(\frac{t-1}{n}\right)^{2(t-1)} \cdot \mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A}^{Type2}) \quad (8)\end{aligned}$$

After obtaining the winning probabilities of $\mathcal{B}$ when dealing with the two types of forgers independently, we can unify them to bound the probability that $\mathcal{B}$ is able to solve his Strong RSA problem with a shrink unforgeability adversary $\mathcal{A}$ as a subroutine.

$$\begin{aligned}&\mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(\mathcal{A}) \\ &\leq 2 \cdot q_h \cdot d \cdot \hat{v} \cdot \left(\frac{n}{t-1}\right)^{2(t-1)} \cdot \mathbf{Adv}_{\mathsf{QR\text{-}SRSA}}(\mathcal{B}) \\ &\quad + d \cdot \hat{v} \cdot \left(\frac{n}{t-1}\right)^{2(t-1)} \cdot (t-1) \cdot \mathbf{Adv}_{\mathsf{DDH}}(\mathcal{D})\end{aligned}$$

It is also not hard to see that if we have a forger with advantage $\mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(T, q_h)$ running in time T, then our Strong RSA adversary would need to run in time $T' = T'' = T + O(t(d\hat{v}\log(n + d\log n) + \log(n!)))$ group operations (*i.e.,* group multiplications and inverses). The additional time needed comes from the hash and signing oracle operations of both Type 1 and Type 2 forger. In particular, the dominant part is the exponentiation costs of the Lagrange interpolation in the signing oracles of the Type 1 and Type 2 scenarios. The complexity of the exponentiation depends on the size of the prime exponents $e_1, \ldots, e_d$. Since we require each prime to be greater than n, each prime number is roughly the size $\log(n + d\log n)$ by the prime number theorem. Thus, the number of squaring operations needed to raise to the exponents is $O(t(d\hat{v}\log(n + d\log n) + \log(n!)))$ since there are t group elements that need to be exponentiated. The additional $\log(n!)$ is due to the extra $n!$ factor in the modified Lagrange interpolation. The simulation in Lemma 1 has a similar time bound due to the use of interpolation. Thus, we conclude the theorem:

$$\begin{aligned}&\mathbf{Adv}^{\mathsf{SU\text{-}CMA}}_{\mathcal{BVS}}(T, q_h) \\ &\leq 2 \cdot q_h \cdot d \cdot \hat{v} \cdot \left(\frac{n}{t-1}\right)^{2(t-1)} \cdot \mathbf{Adv}_{\mathsf{QR\text{-}SRSA}}(T') \\ &\quad + d \cdot \hat{v} \cdot \left(\frac{n}{t-1}\right)^{2(t-1)} \cdot (t-1) \cdot \mathbf{Adv}_{\mathsf{DDH}}(T'')\end{aligned}$$

where $T' = T'' = T + O(t(d\hat{v}\log(n + d\log n) + \log(n!)))$ group operations. $\square$

A.3 Simulation Indistinguishability

The proof above assumes that the Type 1 forger is unable to distinguish our simulation from the real system. Below we show this is the case. More specifically, we tie the forger's ability to distinguish between our simulation and the real system to its ability to distinguish Diffie-Hellman triples from random triples in the group QR_N. To do so, consider the following variant of the decisional Diffie-Hellman problem defined in Section 2.4, to which we refer as the ℓ-DDH problem, where $\mathsf{SafePrimes}$ are the primes p of the form $p = 2p' + 1$ where p' is itself a prime:

$$\begin{aligned}&\mathbf{Adv}_{\ell\text{-}\mathsf{DDH}}(\mathcal{D}) = \\ &|\Pr[\ \mathcal{D}(p, q, g, g^a, \{(g^{b_1}, g^{ab_1}), \ldots, (g^{b_\ell}, g^{ab_\ell})\}) = 1 : \\ &\quad p, q \xleftarrow{\$} \mathsf{SafePrimes} \cap \{0,1\}^\kappa, N \leftarrow pq, g \xleftarrow{\$} QR_N, \\ &\quad a, b_1, \ldots, b_\ell \xleftarrow{\$} \mathbb{Z}_{p'q'} \text{ where} \\ &\quad p' = (p-1)/2, q' = (q-1)/2 \quad] - \\ &\ \Pr[\ \mathcal{D}(p, q, g, g^a, \{(g^{b_1}, g^{r_1}), \ldots, (g^{b_\ell}, g^{r_\ell})\}) = 1 : \\ &\quad p, q \xleftarrow{\$} \mathsf{SafePrimes} \cap \{0,1\}^\kappa, N \leftarrow pq, g \xleftarrow{\$} QR_N, \\ &\quad a, b_1, \ldots, b_\ell, r_1, \ldots, r_\ell \xleftarrow{\$} \mathbb{Z}_{p'q'} \text{ where} \\ &\quad p' = (p-1)/2, q' = (q-1)/2\ ]\ |\end{aligned}$$

Let $\mathbf{Adv}_{\ell\text{-}\mathsf{DDH}}(T) = \max_{\mathcal{D}} \mathbf{Adv}_{\ell\text{-}\mathsf{DDH}}(\mathcal{D})$ where the maximum is taken over all adversaries $\mathcal{D}$ executing in time at most T. It is straightforward to show that $\mathbf{Adv}_{\ell\text{-}\mathsf{DDH}}(T) \leq \ell \cdot \mathbf{Adv}_{\mathsf{DDH}}(T)$ by a standard hybrid argument.

LEMMA 1. *Let X be the probability that a Type 1 forger executing in time T succeeds in forging against the real protocol, and let Y be the probability that this Type 1 forger executing in time T succeeds in forging in our simulation. Then, $|X - Y| \leq \mathbf{Adv}_{(t-1)\text{-}\mathsf{DDH}}(T')$, where $T' = T + O(t(d\hat{v}\log(n + d\log n) + \log(n!)))$ group operations.*

PROOF. Given a Type 1 forger $\mathcal{A}$ that runs in time T, we construct a solver $\mathcal{D}$ for the ℓ-DDH problem in QR_N that runs in time $T + O(t(d\hat{v}\log(n + d\log n) + \log(n!)))$. $\mathcal{D}$ is given an RSA modulus N, and the two strong primes p, q such that $N = pq$. He is also given $g, g^a, \{(g^{b_1}, g^{c_1}), \ldots, (g^{b_{t-1}}, g^{c_{t-1}})\}$, where g is a random generator in QR_N. Intuitively, it should output 1 if each $c_i = ab_i \bmod p'q'$, where $p' = (p-1)/2, q' = (q-1)/2$, and 0 otherwise.

Distinguisher $\mathcal{D}$ simulates the view of the forger as follows. $\mathcal{D}$ assigns the $t-1$ tuples $(g^a, g^{b_1}, g^{c_1}), \ldots, (g^a, g^{b_{t-1}}, g^{c_{t-1}})$ to $t-1$ signing oracles. Without loss of generality, we assume that the first $t-1$ signing oracles $Oracle_1, \ldots, Oracle_{t-1}$ received the tuples. We denote the DDH tuple assigned to a $Oracle_i$ as (A, B_i, C_i), $1 \leq i \leq t-1$.

$\mathcal{D}$ also picks d distinct primes $e_1, \ldots, e_d$ with each $e_i > n$ and computes $sk = \prod_{k=1}^{d} e_k^{-(\hat{\mathbf{v}}[k]+1)} \bmod p'q'$. Suppose that $\mathcal{A}$ is going to make q_h hash oracle queries, $\mathcal{D}$ also randomly selects an index h^* from $\{1, \ldots, q_h\}$. He initializes the forger $\mathcal{A}$ with public key $(N, e_1, \ldots, e_d)$ and prepares to answer his hash and signing oracle queries using the following procedures.

Hash Oracle $\mathcal{H}(c)$:

1. If it is the h^{*th} hash query, it returns A. We refer to this context as c^{**}.
2. Otherwise, it picks a random u and returns g^u. It stores (c, u).

Signing Oracle$_i(\mathbf{v}, c)$:

1. If $1 \leq i \leq t-1$:
   - (a) If $c = c^{**}$, it returns $C_i^{(n!)\prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$.
   - (b) If $c \neq c^{**}$, it looks for the entry (c, u) in the Hash Oracle and returns $B_i^{u\cdot(n!)\prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$.
2. If $t \leq i \leq n$:
   - (a) If $c = c^{**}$, it uses the standard interpolation technique to derive the signature share by using the $t-1$ known shares $C_i^{(n!)\prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$, $i = 1, \ldots, t-1$, and the full signature $A^{sk\cdot(n!)\prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$ to compute the share. Notice here that $\mathcal{D}$ is able to raise to the sk power because he knows the secret key himself. We omit the detail here due to the similarity of the technique and the one in the proof of the theorem.
   - (b) If $c \neq c^{**}$, again it computes the signature share using shares $B_i^{u\cdot(n!)\prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$, $i = 1, \ldots, t-1$, and the full signature $g^{u\cdot sk\cdot(n!)\prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$.

In the end, $\mathcal{D}$ outputs 1 if the forger $\mathcal{A}$ successfully forges a signature and outputs 0 otherwise. Clearly, if each $c_i = ab_i$, then $\mathcal{H}(c^{**}) = g^a$ and the output of $Oracle_i(\mathbf{v}, c^{**})$ is $g^{ab_i(n!)\prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$. Moreover, the distribution of the signing responses for the signing oracles is identical to the real protocol. Then $\mathcal{A}$ forges with probability X.

When each c_i is random, $\mathcal{H}(c^{**}) = g^a$ and $Oracle_i(\mathbf{v}, c^{**})$ outputs $g^{c_i(n!)\prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N = g^{ar_i\cdot(n!)\prod_{k=1}^{d} e_k^{\mathbf{v}[k]}} \bmod N$ for some random r_i. The distribution of the signing responses in our simulation is therefore the same as that provided in the proof of the theorem if we consider r_i to be the signing key for the oracle, in which case $\mathcal{A}$ wins with probability Y.

Thus, we have $\mathbf{Adv}_{(t-1)\text{-DDH}}(\mathcal{D}) = |X - Y|$ and because $\mathcal{D}$ runs in time $T' = T + O(t(d\hat{v}\log(n + d\log n) + \log(n!)))$, we have $\mathbf{Adv}_{(t-1)\text{-DDH}}(T') \geq |X - Y|$. $\square$