ELSEVIER

Theoretical Computer Science 287 (2002) 535–561

Theoretical Computer Science

www.elsevier.com/locate/tcs

Adaptively secure distributed public-key systems

Yair Frankel[a], Philip MacKenzie[b], Moti Yung[c,*]

[a] *Ecash Technologies Inc., 55 Broad Street (22f), New York, NY 1004, USA*
[b] *Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974, USA*
[c] *CertCo Inc., 55 Broad Street (22f), New York, NY 1004, USA*

Received March 2001; accepted May 2001

Abstract

When attacking a distributed protocol, an adaptive adversary is able to determine its actions (e.g., which parties to corrupt) at any time based on its entire view of the protocol including the entire communication history. Proving security of cryptographic protocols against adaptive adversaries is a fundamental problem in cryptography. In this paper, we consider *distributed public-key systems* which are secure against an adaptive adversary. Specifically, we construct distributed discrete-log-based and RSA-based public-key systems secure against an adaptive adversary. We also extend the discrete-log-based systems to have proactive security, that is, security against an (adaptive) mobile adversary that has an upper bound on the number of servers it may corrupt at any one time, but no upper bound on the number of servers it may corrupt over the lifetime of the system.

1. Introduction

Distributed public-key systems involve public/secret key pairs where the secret key is distributively held by some number of servers (using a secret sharing scheme), and a quorum of servers is needed to act on a common input in order to produce a function value (a signature or a cleartext). As long as an adversary does not corrupt a certain threshold of servers the system remains secure (as opposed to centralized cryptosystems in which the compromise of a single entity breaks the system). Function sharing (Threshold) systems were presented in [12–14]. Robust function sharing systems, in which the function can be evaluated correctly even if the adversary causes the servers it controls to misbehave arbitrarily, were presented in [28, 22, 27]. Proactively secure systems, which maintain security and robustness against mobile adversaries [37] that

* Corresponding author.
E-mail addresses: yfrankel@cs.columbia.edu (Y. Frankel), philmac@research.bell-labs.com (P. MacKenzie), moti@cs.columbia.edu (M. Yung).

0304-3975/02/$ - see front matter
PII: S0304-3975(01)00260-2

may corrupt different sets of servers over time (possibly all servers at one time or another), were presented in [32, 21, 8, 20, 39]. Constructions of these systems are required to be efficient (e.g., they should not involve generic *secure function evaluation* which is assumed impractical [12]). The current trend for specific efficient solutions is reviewed in [30, 26].

All the previous protocols for distributed cryptosystems have been proven secure against a static adversary, i.e., an adversary that decides its actions (and specifically, its corruption strategy) before the execution of the protocol. We say a protocol is *adaptively secure* if it is secure against an adaptive adversary. Adaptive security is a fundamental problem in cryptography, and has been recently considered in the context of multi-party computations (secure function evaluation), initially where parties erase some of their information [4], and later even when they do not necessarily do so [5]. An adaptively secure oblivious transfer protocol was given in [3].

In this paper, we examine the problem of obtaining adaptively secure distributed public-key systems. Adaptive security for distributed public-key systems has also been considered recently in Canetti et al. [6], which is extended by Jarecki and Lysyanskaya [35], where non-optimally resilient protocols for discrete-log based key generation and threshold function application are proven secure against adaptive adversaries. They sketch how to construct corresponding optimally resilient protocols, and also how to construct a specific l-out-of-l RSA system). The work in this paper was developed independently from the work of Canetti et al. [6], and deals with both discrete-log-based (DL-based) [15] and RSA-based [40] systems. In a companion paper [25], this work is extended to proactive maintenance of RSA-based systems based on polynomial sharing.

The major difficulty in proving the security of protocols against adaptive adversaries is being able to efficiently and consistently simulate the view of an adversary which may corrupt parties dynamically, depending on some internal, and possibly bizarre, strategy. In particular, when a party is corrupted, its (simulated) internal state must be consistent with the current view of the adversary. However, the simulator does not know the true secret keys, and is not able to determine the true internal states of all parties simultaneously. In fact, in most protocols the simulator is not able to determine the true internal states of more than half of the parties in the system. Therefore, the simulator may have difficulty producing a consistent internal state for an arbitrary set of parties that is corrupted. (Recall that the number of possible sets is *exponential* in the total number of parties.) The technique of backtracking (often used in zero-knowledge proofs) is problematic since after backtracking the view of the adversary will most likely change, and the adversary may then corrupt a different set of parties based on its internal strategy.

Remark In real systems, there is no reason why an adversary could not corrupt parties dynamically according to any strategy it chooses, and thus adaptive security is a very natural and important concept.

In distributed public-key systems, the problem of adaptive security is exacerbated by the fact that there is generally public function and related publicly committed robustness

information available to anyone, which as discussed above, needs to be consistent with internal states of parties which get corrupted. This is the main cause of difficulties in the proof of security.

Our contributions and techniques. We give a new set of techniques that can be used to construct distributed DL-based and RSA-based public-key systems with adaptive security. Since the simulation-based proofs of the earlier techniques fail against an adaptive adversary, we have to employ new ideas. The driving "meta idea" is to develop techniques that assure, in spite of the exponential set of behaviors of the adversary, that the adversary can only disrupt the simulation with polynomial probability. This argument will assure simulatability and thus a proof of security. The basic principle is based on the notion of a *faking server*. The simulator exploits the actions of this server to assure that the view is simulatable while not knowing the secret key. This server is chosen at random and its public actions are indistinguishable from an honest server to the adversary. We have to backtrack the simulation *only if* the adversary corrupts this special server. Since there is only one faking server, and since regardless of its corruption strategy, the adversary has a polynomial chance (at least $1/(t+1)$) of not corrupting this one server, we will be able to complete the simulation in expected polynomial time.

We employ non-binding encryption and develop the notion of *detached commitments*, that is, commitments that can be used to ensure correct behavior of servers (i.e., robustness), yet have no *hard attachment* to the secret key, thus allowing an efficient simulation. First we show how to work with these detached commitments, e.g., using *function representation transformations* like *poly-to-sum* and *sum-to-poly* (which we build based on Frankel et al. [20]). Next we show how to maintain robustness by constructing simulatable *soft attachments* from these detached commitments to the secret-key-based *function applications*. The soft attachments are constructed using efficient zero-knowledge proofs-of-knowledge.[1] The protocols we give for these zero-knowledge proofs-of-knowledge are similar to Cramer et al. [11], but are novel in the following ways: (1) there is one *setup* protocol for all the *proof* protocols, which allows concurrency in the proof protocols, and (2) the setup and proof protocols are not as tightly related (i.e. the setup does not prove knowledge of a commitment based on exactly what the proof protocol is trying to prove) but still achieve statistical zero-knowledge (no reliance on computational assumptions). We believe these ZK-proof techniques will be useful in developing future threshold cryptosystems.

We note that the detached commitment methodology is crucial against an adaptive adversary. It allows the security by simulation to deal with simulation of a message signing (or decryption) without rewinding the simulation beyond the current signing (decryption). It enables to translate an adaptive adversary who chooses messages and

[1] These are actually computationally sound proofs, or *arguments*, but we will simply use the term proof.

corruption of servers to an adversary against the centralized system with the same choice of messages.[2]

Our techniques maintain "optimal resilience", namely, the protocols can withstand any minority of misbehaving servers (t faults out of l servers while $l \geqslant 2t+1$ is allowed). Our main results are:

Theorem 1. *There exists an adaptively secure proactive DL-based optimal-resilient* (t,l)*-threshold public-key system.*

Theorem 2. *There exists an adaptively secure robust RSA-based optimal-resilient* (t,l)*-threshold public-key system.*

In a companion paper [25], we extend this work to proactive-maintenance of RSA-based systems, which requires additional techniques.

2. Model and definitions

Our system consists of l servers $\mathcal{S}=\{S_1,\ldots,S_l\}$. A server is *corrupted* if it is controlled by the adversary. When a server is corrupted, we assume without loss of generality that the adversary sees all the information currently on that server. On the other hand, the system should not open secrets of unavailable servers. Namely, we separate availability faults from security faults (and do not cause security exposures due to unavailability). We assume that all un-corrupted servers receive all messages that are broadcast, and may retrieve the information from messages encrypted with a public key, if they know the corresponding private key. Our communication model is similar to Herzberg et al. [33]. All participants communicate via an authenticated bulletin board [9] in a synchronized manner.

The adversary. Our threshold schemes assume a *stationary* adversary which stays at corrupt servers, i.e. once a server is corrupted, it stays corrupted. Our proactive schemes assumes a *mobile* adversary as defined in [37, 32]. That is, time is divided into alternating operational periods (where function evaluation takes place) and update periods (where refreshing of key shares takes place), and an adversary may corrupt different servers in different time periods, with servers possibly being refreshed or rebooted (i.e. they become uncorrupted) during update periods.

In all schemes we assume the adversary is *t-restricted*; namely it can, during the lifetime of the system (or during any period, for proactive systems), corrupt at most t servers. The actions of an adversary at any time may include submitting messages to

[2] The related work in Canetti et al. [6] does not use this methodology. From their sketch of the threshold t-out-of-l RSA system, it seems that the following is needed in their system: to be able to simulate they need an l-out-of-l update of keys after each signature. This is not a threshold scheme due to availability requirements (which needs all servers to be available for each signature). Also, it seems that it may be hard to prove security of history-dependent signature schemes in such a case.

the system to be signed, corrupting servers, and broadcasting arbitrary information on the communication channel. The adversary is *adaptive*; namely it is allowed to base its actions not only on previous function outputs, but on all the information that it has previously obtained during the execution of the protocol.

Remark When proving security the adversary is translated into an *adaptive* adversary which makes the same adaptive message challenges against the centralized system (where the key is held by a centralized server and no protocol is performed to answer the challenge).

2.1. DL-based and RSA-based systems

Next we give information on the basics of DL-based and RSA-based systems. It includes, among other information, details of the variants of secret sharing schemes that are used, such as Shamir threshold secret sharing and Pedersen unconditionally secure threshold verifiable secret sharing ((t,l)-US-VSS) over known groups, along with secret sharing and unconditionally secure threshold verifiable secret sharing (INT-(t,l)-US-VSS) over the integers, with check shares computed modulo an RSA modulus.

2.1.1. Basics for DL-based systems

In these systems, we assume that p and q are two primes such that $p=mq+1$ (in many cases $m=2$ is used), and that g is an elements of Z_p^* of order q, so $g^q \equiv 1 \bmod p$. Various signature schemes have been developed based on the intractability of computing discrete logs, which formally is stated as follows:

DLP Assumption. Let k be the security parameter. The DLP assumption is as follows. Given primes p and q as above with $|p|=k$, and given an element $g \in Z_p^*$ of order q, and the group G_g generated by g in Z_p^*, for any polynomial-time algorithm A, $\Pr[g^x \equiv y \bmod p : y \in_R G_g, x \leftarrow A(1^k, p, q, g, y)]$ is negligible.

We use various sharing techniques. We assume the reader is familiar with Shamir (t,l)-threshold polynomial secret sharing [42]. We will use the polynomial interpolation formula explicitly, so we will describe it here. For a t-degree polynomial $v(x)$, and a set $\Lambda=\{i_1,\ldots,i_{t+1}\}$ of size $t+1$, $v(0)$ can be computed using polynomial interpolation. Define $z_{i,\Lambda}=\prod_{j\in\Lambda\setminus\{i\}}(i-j)^{-1}(0-j)$. Then $v(0)=\sum_{i\in\Lambda} v(i)z_{i,\Lambda}$.

We now describe an unconditionally secure (t,l)-VSS ((t,l)-US-VSS) due to Pedersen [38] where secrets are drawn from Z_q and verification shares are computed in Z_p. We assume an h in G_g is generated in some way so that the discrete log of h relative to g is unknown (this can be assured via proper initialization).

The protocol begins with two (t,l)-threshold polynomial secret sharings, sharing secrets $s,s' \in Z_q$. Let $a(x)=\sum_{j=0}^{t} a_j x^j$ be the random polynomial used in sharing s and let $a'(x)=\sum_{j=0}^{t} a'_j x^j$ be the random polynomial used in sharing s'. For all i, S_i receives shares $s_i=a(i)$ and $s'_i=a'(i)$. (We refer to the pair $(a(i),a'(i))$ as *dual-share* i.)

Also, the *verification shares* $\{\alpha_j(=g^{a_j}h^{a'_j})\}_{0\leqslant j\leqslant t}$, are published.[3] Say *check share* $A_i=\prod_{j=0}^{t}\alpha_j^{i^j}$. S_i can verify the correctness of his shares by checking that $A_i=g^{s_i}h^{s'_i}$. Say s and s' are the shares computed using Lagrange interpolation from a set of $t+1$ shares that passed the verification step. If the dealer can reveal different secrets $\hat{s}$ and $\hat{s}'$ that also correspond to the zero coefficient verification share, then the dealer can compute a discrete log of h relative to g.

2.1.2. Basics for RSA-based systems

RSA-based systems rely on the intractability of computing RSA inverses, and hence, the intractability of factoring products of two large primes. Let k be the security parameter. Let key generator GE define a family of RSA functions to be $(e,d,N)\leftarrow GE(1^k)$ such that N is a composite number $N=P*Q$ where P,Q are prime numbers of $k/2$ bits each. The exponent e and modulus N are made public while $d\equiv e^{-1} \bmod \lambda(N)$ is kept private.[4] The *RSA encryption function* is public, defined for each message $M\in Z_N$ as: $C=C(M)\equiv M^e \bmod N$. The *RSA decryption function* (also called signature function) is the inverse: $M=C^d \bmod N$. It can be performed by the owner of the private key d. Formally, the RSA Assumption is stated as follows:

RSA Assumption. Let k be the security parameter. Let key generator GE define a family of RSA functions (i.e., $(e,d,N)\leftarrow GE(1^k)$ is an RSA instance with security parameter k). For any probabilistic polynomial-time algorithm A, $\Pr[u^e\equiv w \bmod N:(e,d,N)\leftarrow GE(1^k); w\in_R\{0,1\}^k; u\leftarrow A(1^k,w,e,N)]$ is negligible.

Next we describe variants of Shamir secret sharing and Pedersen VSS that we use in RSA-based systems. They differ in that operations on the shares are performed over the integers, instead of in a modular subgroup of integers.

(t,l)-secret sharing over the integers (INT-(t,l)-SS) [20]. This is a variant of Shamir secret sharing [42]. Let $L=l!$ and let m be a positive integer. For sharing a secret $s\in[0,mK]$ (and K the size of an interval over the integers), a random polynomial $a(x)=\sum_{j=0}^{t}a_jx^j$ is chosen such that $a_0=L^2s$, and each other $a_j\in_R\{0,L,2L,\ldots,mL^3K^2\}$.[5] Each shareholder $i\in\{1,\ldots,l\}$ receives a secret share $s_i=a(i)$, and verifies[6] that (1) $0\leqslant s_i\leqslant mL^3K^2l^{t+1}$, and (2) L divides s_i. Any set Λ of cardinality $t+1$ can compute s using Lagrange interpolation.

Unconditionally-secure (t,l)-VSS over the integers (INT-(t,l)-US-VSS). This is a variant of Pedersen unconditionally-secure (t,l)-VSS [38], and is slightly different than

[3] In DL-based systems, we implicitly assume all verification operations are performed in Z_p^*.

[4] $\lambda(N)=\mathrm{lcm}(P-1,Q-1)$ is the smallest integer such that any element in Z_N^* raised by $\lambda(N)$ is the identity element. RSA is typically defined using $\phi(N)$, the number of elements in Z_N^*, but it is easy to see that $\lambda(N)$ can be used instead. We use it because it gives an explicit way to describe an element of maximal order in Z_N^*. Note that $\phi(N)$ is a multiple of $\lambda(N)$, and that knowing any value which is a multiple of $\lambda(N)$ implies breaking the system.

[5] We note that in our RSA-based systems, L^2s is actually the secret component of the RSA secret key, which when added to a public leftover component (in $[0,L^2-1]$), forms the RSA secret key.

[6] These tests only verify the shares are of the correct form, not that they are correct polynomial shares.

the version in [24]. Let N be an RSA modulus and let g and h be generators whose discrete log modulo N with respect to each other is unknown. The protocol begins with two (t,l)-secret sharings over the integers, the first sharing secret s with $m=1$, and the second sharing s' with $m=NK$. Note that $s\in[0,K]$ and $s'\in[0,NK^2]$. Let $a(x)=\sum_{j=0}^{t}a_jx^j$ be the random polynomial used in sharing s and let $a'(x)=\sum_{j=0}^{t}a'_jx^j$ be the random polynomial used in sharing s'. For all i, S_i receives shares $s_i=a(i)$ and $s'_i=a'(i)$. (We refer to the pair $(a(i),a'(i))$ as *dual-share i*.) Also, the *verification shares* $\{\alpha_j(=g^{a_j}h^{a'_j})\}_{0\leqslant j\leqslant t}$, are published.[7] Say *check share* $A_i=\prod_{j=0}^{t}\alpha_j^{i^j}$. S_i can verify the correctness of his shares by checking that $A_i=g^{s_i}h^{s'_i}$. Say s and s' are the shares computed using Lagrange interpolation from a set of $t+1$ shares that passed the verification step. If the dealer can reveal different secrets $\hat{s}$ and $\hat{s}'$ that also correspond to the zero coefficient verification share, then the dealer can compute an α and β such that $g^{\alpha}\equiv h^{\beta}$, which implies factoring, and thus breaking the RSA assumption (see Lemma 3).

Looking ahead, we will need to simulate an INT-(t,l)-US-VSS. Using Lemma 2, we can do this by constructing a random polynomial over an appropriate simulated secret (e.g., a random secret, or a secret obtained as a result of a previously simulated protocol) in the zero coefficient, and a random companion polynomial with a totally random zero coefficient. Note that the β value in the lemma will correspond to K, and the γ value in the lemma will correspond to the discrete log of g with respect to h, which is less than N. The probability of distinguishing a real VSS from the simulated VSS will be $(4t+2)/K$, which is exponentially small if the range of secrets K is exponentially large.

2.2. Distributed public-key systems

We will say that the secret key x is shared among the servers, and each server S_i holds share x_i. The public key associated with x will be called y. The operations performed could include cryptographic function application, or proactive updating of the shares. We call these operations DISTAPPLY, and DISTUPDATE, respectively. We also consider operation VERIFY, which verifies the output of DISTAPPLY.[8] We say a (t,l)-*threshold system* is a threshold system with l servers that is designed to withstand a t-restricted adaptive adversary. We similarly may have (t,l)-*proactive systems*.

Next we give formal definitions for distributed public-key systems.

Definition 1 (*Robustness of a threshold system*). A (t,l)-threshold public-key system S is robust if for any polynomial-time t-restricted adaptive stationary adversary $\mathscr{A}$, with

[7] In RSA-based systems, we implicitly assume all verification operations are performed in Z_N^*.

[8] For encryption schemes like RSA, the DISTAPPLY operation is RSA decryption, and the VERIFY operation is simply RSA-encryption of the output of DISTAPPLY and verification that it equals the input. For encryption schemes like El-Gamal, the VERIFY operation may not be polynomial time. Note that the security and robustness definitions still make sense in both of these cases.

all but negligible probability, for each input m which is submitted to the DISTAPPLY protocol the resulting output s passes $VERIFY(y,m,s)$.

Definition 2 (*Security of a threshold system*). A (t,l)-threshold public-key system S is secure if for any polynomial-time t-restricted adaptive stationary adversary $\mathscr{A}$, after polynomially-many DISTAPPLY protocols performed on given values, given a new value m and the view of $\mathscr{A}$, the probability of being able to produce an output s that passes $VERIFY(y,m,s)$ is negligible.

Definition 3 (*Robustness of a proactive system*). A (t,l)-proactive public-key system S is robust if for any polynomial-time t-restricted adaptive mobile adversary $\mathscr{A}$, with all but negligible probability, after polynomially-many DISTUPDATE protocols for each input m which is submitted to the DISTAPPLY protocol during an operational period, the resulting output s passes $VERIFY(y,m,s)$.

Definition 4 (*Security of a proactive system*). A (t,l)-proactive public-key system S is secure if for any polynomial-time t-restricted adaptive mobile adversary $\mathscr{A}$, after polynomially-many DISTAPPLY protocols performed during operational periods on given values, and polynomially-many DISTUPDATE protocols, given a new value m and the view of $\mathscr{A}$, the probability of being able to produce an output s that passes $VERIFY(y,m,s)$ is negligible.

Remark. The way m and the prior inputs are chosen in Definitions 2 and 4 corresponds to the type of adversary that would be considered in the corresponding non-distributed scheme. For instance, for a signature scheme that is secure against existential forgery under adaptive chosen message attack, the value m and the prior inputs may be chosen by the adversary in the distributed scheme. On the other hand, for a decryption scheme like (standard) RSA, we assume in the distributed decryption scheme that the prior inputs are chosen randomly, and the ciphertext m to be decrypted is given as input to the adversary. In this paper we do not deal with stronger notions of choice of decryption challenges (see [43, 7]).

3. Techniques

The main problem with proving security and robustness against adaptive adversaries is that public values such as ciphertexts and commitments are linked to actual cleartext values in an undeniable fashion. To detach ciphertexts from their cleartext values, we simply employ semantically secure non-committing encryption [4]. In fact, our security proofs first assume perfectly secret channels and then add the above (a step we omit here), using by now standard arguments about replacement of secure channels with encryption.

A more involved issue concerns the commitments. We know that the collection of techniques needed to underly distributed public-key systems include: distributed

representation methods (polynomial sharing, sum (additive) sharing), representation transformers which move between different ways to represent a function (poly-to-sum, sum-to-poly) as well as a set of elementary distributed operations (add, multiply, invert). For example, the *poly-to-sum* protocol is executed by $t+1$ servers at a time, and transforms $(t+1)$-out-of-l polynomial-based sharings to $(t+1)$-out-of-$(t+1)$ additive sharings. We need to have such techniques (motivated by Frankel et al. [24, 20]) which are secure and robust against adaptive adversaries. In the coming subsections, we represent the basic techniques to manipulate representation of shared function.

We will rely on new zero-knowledge proof techniques (see Appendix A), as well as on shared representation of secrets as explained in Section 2.1.

The notation *2poly* refers to a polynomial and its companion polynomial shared with (t,l)-US-VSS (which is *unconditionally secure VSS*) (or INT-(t,l)-US-VSS (which is the same but over the integers)).

The notation *2sum* refers to two additive sharings, with check shares that contain both additive shares of a server (similar to the check shares in (t,l)-US-VSS). In describing the DL-based protocols, unless otherwise noted we will assume multiplication is performed mod p and addition (of exponents) is performed mod q. In describing the RSA-based protocols, unless otherwise noted we will assume multiplication is performed mod N and addition (of exponents) is performed over the integers (i.e., not "mod" anything).

3.1. 2poly-to-2sum

The goal of 2poly-to-2sum is to transform t-degree polynomials $a(\ )$ and $a'(\ )$ used in (t,l)-US-VSS into $t+1$ additive shares for each secret $a(0)$ and $a'(0)$, with corresponding check shares. The idea is to perform interpolation.[9] The DL-based scheme shown in Fig. 1 does not actually require any communication, since all check shares

1. *Initial configuration*: (t,l)-US-VSS (parameters: (p,q,g,h)) with t-degree polynomials $a(\ )$ and $a'(\ )$, and a set Λ of $t+1$ server indices. For all $i \in \Lambda$, recall S_i holds shares s_i and s'_i with corresponding check share $A_i = g^{s_i} h^{s'_i}$.
2. S_i computes additive shares $b_i = s_i z_{i,\Lambda}$ and $b'_i = s'_i z_{i,\Lambda}$.
3. Every server computes the check shares $B_i = g^{b_i} h^{b'_i} = A_i^{z_{i,\Lambda}}$ for all $i \in \Lambda$. (Note that there is no communication, since the additive shares can be computed individually by each shareholder, and all check shares can be computed from publicly available verification shares.)

Fig. 1. 2poly-to-2sum: DL-based scheme.

[9] As opposed to [20], in which poly-to-sum also performed a rerandomization of the additive shares, we have separate protocols for (1) converting a polynomial sharing into an additive sharing and (2) rerandomizing an additive sharing, since sometimes we do not need to do a rerandomization of additive shares.

1. *Initial configuration*: INT-(t,l)-US-VSS (parameters: (N,g,h)) with t-degree polynomials $a(\ )$ and $a'(\ )$, and a set Λ of $t+1$ server indices. For all $i\in\Lambda$, recall S_i holds shares s_i and s'_i with corresponding check share $A_i=g^{s_i}h^{s'_i}$.
2. For all $i\in\Lambda$, S_i computes the additive shares $b_i=s_iz_{i,\Lambda}$ and $b'_i=s'_iz_{i,\Lambda}$ and publishes $B_i=g^{b_i}h^{b'_i}=A_i^{z_{i,\Lambda}}$.
3. All servers verify B_i for all $i\in\Lambda$ using $(A_i)^{V_{i,\Lambda}}\equiv(B_i)^{V'_{i,\Lambda}}$ where $V_{i,\Lambda}=\prod_{j\in\Lambda\setminus\{i\}}(0-j)$ and $V'_{i,\Lambda}=\prod_{j\in\Lambda\setminus\{i\}}(i-j)$. If the verification for a given B_i fails, each server broadcasts a (Bad,i) message and quits the protocol.

Fig. 2. 2poly-to-2sum: RSA-based scheme.

1. *Initial configuration*: (Parameters (p,q,g,h)) There is a set Λ of $t+1$ server indices. For all $i\in\Lambda$, S_i holds additive dual-share (b_i,b'_i), with corresponding check share $B_i=g^{b_i}h^{b'_i}$.
2. For all $i\in\Lambda$, S_i chooses $r_{i,j}\in_R Z_q$ and $r'_{i,j}\in_R Z_q$ for $j\in\Lambda\setminus\{i\}$.
3. For all $i\in\Lambda$, S_i sets $r_{i,i}=b_i-\sum_{j\in\Lambda\setminus\{i\}}r_{i,j}$ and $r'_{i,i}=b'_i-\sum_{j\in\Lambda\setminus\{i\}}r'_{i,j}$.
4. For all $i\in\Lambda$, S_i privately transmits $r_{i,j}$ and $r'_{i,j}$ to all S_j for $j\in\Lambda\setminus\{i\}$.
5. For all $i\in\Lambda$, S_i publishes $R_{i,j}=g^{r_{i,j}}h^{r'_{i,j}}$ for $j\in\Lambda\setminus\{i\}$.
6. All servers can compute $R_{i,i}=B_i/\prod_{j\in\Lambda\setminus\{i\}}R_{i,j}$ for all $i\in\Lambda$.
7. For all $j\in\Lambda$, S_j verifies $R_{i,j}\equiv g^{r_{i,j}}h^{r'_{i,j}}$. If the verification fails, S_j broadcasts an (Accuse,i,$R_{i,j}$) message, to which S_i responds by broadcasting $r_{i,j}$ and $r'_{i,j}$. If S_i does not respond, or $R_{i,j}\not\equiv g^{r_{i,j}}h^{r'_{i,j}}$ (which all servers can now test), then each server broadcasts a (Bad,i) message and quits the protocol.
8. For all $j\in\Lambda$, S_j computes $d_j=\sum_{i\in\Lambda}r_{i,j}$, $d'_j=\sum_{i\in\Lambda}r'_{i,j}$, and $D_j=\prod_{i\in\Lambda}R_{i,j}$.

Fig. 3. 2sum-to-2sum: DL-based scheme.

can be computed from public information. The RSA-based scheme shown in Fig. 2 is similar, but requires S_i to broadcast the check share, since it cannot be computed by every server. We note that in Step 2 each s_i and s'_i is a multiple of L, so S_i can actually compute b_i and b'_i over the integers.

3.2. 2sum-to-2sum

The goal of 2sum-to-2sum is to randomize additive dual-shares (most likely obtained from a 2poly-to-2sum) and update the corresponding check shares. The DL-based scheme is in Fig. 3 and the RSA-based scheme in Fig. 4.

3.3. 2sum-to-1sum

The goal of 2sum-to-1sum is to reveal check shares corresponding to the first half of additive dual-shares, and prove they are correct. These proofs form the *soft attachments* from the information-theoretically secure verification shares to the computationally

1. *Initial configuration*: (Parameters: (N,g,h)) There is a set Λ of $t+1$ server indices. For all $i \in \Lambda$, S_i holds additive dual-share (b_i, b_i'), with corresponding check share $B_i = g^{b_i} h^{b_i'}$.
2. For all $i \in \Lambda$, S_i chooses $r_{i,j} \in_R Z_N$ and $r'_{i,j} \in_R Z_N$, for $j \in \Lambda \backslash \{i\}$.
3. For all $i \in \Lambda$, S_i sets $r_{i,i} = b_i - \sum_{j \in \Lambda \backslash \{i\}} r_{i,j}$ and $r'_{i,i} = b_i' - \sum_{j \in \Lambda \backslash \{i\}} r'_{i,j}$.
4. For all $i \in \Lambda$, S_i privately transmits $r_{i,j}$ and $r'_{i,j}$ to all S_j for $j \in \Lambda \backslash \{i\}$.
5. For all $i \in \Lambda$, S_i publishes $R_{i,j} = g^{r_{i,j}} h^{r'_{i,j}}$ for $j \in \Lambda \backslash \{i\}$.
6. All servers can compute $R_{i,i} = B_i / \prod_{j \in \Lambda \backslash \{i\}} R_{i,j}$ for all $i \in \Lambda$.
7. For all $j \in \Lambda$, S_j verifies that each $r_{i,j}$ and $r'_{i,j}$ received is in the correct range and that $R_{i,j} \equiv g^{r_{i,j}} h^{r'_{i,j}}$. If the verification fails, S_j broadcasts an (Accuse,i,$R_{i,j}$) message, to which S_i responds by broadcasting $r_{i,j}$ and $r'_{i,j}$. If S_i does not respond, $r_{i,j}$ or $r'_{i,j}$ is not in the correct range, or $R_{i,j} \not\equiv g^{r_{i,j}} h^{r'_{i,j}}$ (which all servers can now test), then each server broadcasts a (Bad,i) message and quits the protocol.
8. For all $j \in \Lambda$, S_j computes $d_j = \sum_{i \in \Lambda} r_{i,j}$, $d_j' = \sum_{i \in \Lambda} r'_{i,j}$, and $D_j = \prod_{i \in \Lambda} R_{i,j}$.

Fig. 4. 2sum-to-2sum: RSA-based scheme.

1. *Initial configuration*: Parameters (p,q,g,h). There is a set Λ of $t+1$ server indices. For all $i \in \Lambda$, S_i holds additive dual-share (d_i, d_i'), with corresponding check share $D_i = g^{d_i} h^{d_i'}$. Also, all servers S_i with $i \in \Lambda$ have performed a ZK-proof-setup protocol ZKSETUP-DL(p,q,g,h) with all other servers.
2. For all $i \in \Lambda$, S_i broadcasts $E_i = g^{d_i}$.
3. For all $i \in \Lambda$, S_i performs a ZK-proof of knowledge ZKPROOF-DL-REP (p,q,g,h,E_i,h,D_i) with all other servers. Recall that this is performed over a broadcast channel so all servers can check if the ZK-proof was performed correctly.
4. If a server detects that for some $i \in \Lambda$, S_i fails to perform the ZK-proof correctly, that server broadcasts a message (Bad,i) and quits the protocol

Fig. 5. 2sum-to-1sum: DL-based scheme.

secure check shares that must correspond to the actual secret. The DL-based scheme is shown in Fig. 5 and the RSA-based scheme is shown in Fig. 6.

3.4. 2sum-to-2poly

The protocol is given in Fig. 7.

4. Protocols

To begin, we present protocols for DISTAPPLY (threshold cryptographic function application), and DISTUPDATE (proactive maintenance of key shares), for both DL-based and

1. *Initial configuration*: Parameters (N, e, g, h). There is a set Λ of $t+1$ server indices. For all $i \in \Lambda$, S_i holds additive dual-share (d_i, d_i'), with corresponding check share $D_i = g^{d_i} h^{d_i'}$. Also, all servers S_i with $i \in \Lambda$ have performed a ZK-proof-setup protocol ZKSETUP-RSA(N, e, g) with all other servers.
2. For all $i \in \Lambda$, S_i broadcasts $E_i = m^{d_i}$, where m is the message to be signed, or more generally, the value to which the cryptographic function is being applied.
3. For all $i \in \Lambda$, S_i performs a ZK-proof of knowledge ZKPROOF-IF-REP $(N, e, g, m, E_i, h, D_i)$ with all other servers. Recall that this is performed over a broadcast channel so all servers can check if the ZK-proof was performed correctly.
4. If a server detects that for some $i \in \Lambda$, S_i fails to perform the ZK-proof correctly, that server broadcasts a message (Bad,i) and quits the protocol.

Fig. 6. 2sum-to-1sum: RSA-based scheme.

1. *Initial configuration*: Parameters (p, q, g, h). There is a set Λ of server indices. For all $i \in \Lambda$, S_i holds additive dual-share (d_i, d_i'), with corresponding check share $D_i = g^{d_i} h^{d_i'}$.
2. For each $i \in \Lambda$, S_i shares d_i and d_i' using (t, l)-US-VSS, say with polynomials $v_i(\ )$ and $v_i'(\ )$.
3. S_j computes the sums $v(j) = \sum_{i \in \Lambda} v_i(j)$ and $v'(j) = \sum_{i \in \Lambda} v_i'(j)$. The verification shares for $v(\ )$ and $v'(\ )$ can be computed from the verification shares for $v_i(\ )$ and $v_i'(\ )$, for $i \in \Lambda$.
4. If a verification fails for the (t, l)-US-VSS from S_i, each server broadcasts (Bad, i). When 2sum-to-2poly is used for proactive maintenance, the servers quit the protocol.

Fig. 7. 2sum-to-2poly: DL-based scheme.

RSA-based systems. We then present the complete proactive threshold cryptosystems for both DL-based and RSA-based systems (assuming the keys are already distributed). The security and robustness of these protocols are proven in Section 5.3.

4.1. DL-based threshold function application

Here we consider any DL-based (t, l)-threshold function application protocol that works by (1) constructing a verifiable additive representation of the secret x over $t+1$ servers with check shares over g, (2) finishing the function application with those $t+1$ servers (we will call this the *additive application* step), if there is no misbehavior, and (3) going back to step (1) if misbehavior is detected, discarding servers which have misbehaved and using $t+1$ remaining servers. We assume that there is a simulator

1. *Initial configuration*: (t,l)-US-VSS (parameters: (p,q,g,h)) with t-degree polynomials $a(\ )$ and $a'(\ )$. Also, each server maintains a list $\mathcal{G}$ of server indices for servers that have not misbehaved (i.e., they are considered good). A message m needs to be signed.
2. A set $\Lambda \subseteq \mathcal{G}$ with $|\Lambda| = t+1$ is chosen in some public way.
3. 2poly-to-2sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 2.
4. 2sum-to-2sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 2.
5. 2sum-to-1sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 2.
6. The additive application step of the signature protocol is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 2.
7. All values created during the signing protocol for m are erased.

Fig. 8. DL-based function application protocol.

for the additive application step for a message m which can simulate the step with inputs consisting of $t+1$ additive shares with $t+1$ check shares, where at most one additive share does not correspond to its check share, and a signature on m. The simulator fails only if the *faking server* (the one containing the unmatched share and check share) is corrupted, and otherwise provides a view to the adversary which is perfectly indistinguishable from the view the adversary would have in the real protocol. Most robust threshold DL-based protocols against static adversaries, like AMV-Harn [1, 31] and El-Gamal decryption [18] work this way. For an example with AMV-Harn signatures, see [34]. We show how to use this technique for static adversaries to construct a protocol that withstands an adaptive adversary. Specifically, we use a (t,l)-US-VSS representation to store the secret, and for function application we use 2poly-to-1sum (shorthand for the concatenation of 2poly-to-2sum, 2sum-to-2sum, and 2sum-to-1sum) to construct the verifiable additive representation of the secret. We call this the *DL-based function application protocol*. The protocol is given in Fig. 8.

4.2. RSA-based threshold function application

We define *RSA-based function application protocols* analogously to DL-based function application protocols. The main change is that the version of VSS over the integers (INT-(t,l)-US-VSS) should be used. (An example of this type of protocol for RSA signature and decryption functions is given in Frankel et al. [20].) This allows the simulator to construct a view for the adversary which is statistically indistinguishable (as opposed to perfectly indistinguishable in the DL-based protocols) from the view the adversary would have in the real protocol. We also shortcut the additive

1. The dealer generates an RSA public/private key (N,e,d), and computes public value x^* and secret value x such that $d \equiv x^* + L^2 x \bmod \phi(N)$, as in [21].[a] Then the dealer generates generators $g,h \in_R Z_N^*$, $x' \in_R Z_{N^3}$, and an INT-(t,l)-US-VSS (with $K=N$, i.e., the range of the secret is assumed to be $[0,N]$) on secrets x,x' with parameters (N,g,h).
2. Each (ordered) pair of servers (S_i,S_j) performs ZKSETUP-RSA$_{S_i,S_j}(N,e,g,h)$.
3. Each server maintains a list $\mathcal{G}$ of server indices for servers that have not misbehaved (i.e., they are considered good).
4. When a message m needs to be signed, the following DISTAPPLY protocol is run:
   (a) A set $\Lambda \subseteq \mathcal{G}$ with $|\Lambda| = t+1$ is chosen in some public way.
   (b) 2poly-to-2sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 4a.
   (c) 2sum-to-2sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 4a.
   (d) 2sum-to-1sum is run. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 4a. If there is no misbehavior, the signature on m can be computed from the partial signatures generated in this step.
   (e) All values created during the signing protocol for m are erased.

[a] Recall that x^* is computed using only the public values N,e,L.

Fig. 9. Basic RSA-based threshold protocol [21].

application step, and simply form the partial RSA signatures in the 2sum-to-1sum step.

The protocol is given in Fig. 9. (Because we will not discuss proactive RSA systems, we actually present here a complete description of a basic RSA-based threshold system that includes the function application protocol.)

4.3. DL-based proactive maintenance

For DL-based proactive maintenance, we perform an update by running 2poly-to-2sum on the secret polynomials, and then 2sum-to-2poly. After 2sum-to-2poly, each server erases all previous share information, leaving just the new polynomial shares. If there is misbehavior by a server in either protocol, the procedure is restarted with new participants (here restarts do not introduce statistical biases and do not reduce the protocol's security). The protocol is given in Fig. 10. The complete proactive system is outlined in Fig. 11.

1. *Initial configuration*: (t,l)-US-VSS (parameters: (p,q,g,h)) with t-degree polynomials $a(\ )$ and $a'(\ )$
2. Each server broadcasts what it believes h to be, and the correct h is determined by a majority vote.
3. Each (ordered) pair of servers (S_i,S_j) performs ZKSETUP-$\mathrm{DL}_{S_i,S_j}(p,q,g,h)$.
4. Each server maintains a list $\mathscr{G}$ of server indices for servers that have not misbehaved (i.e., they are considered good).
5. A set $\Lambda\subseteq\mathscr{G}$ with $|\Lambda|=t+1$ is chosen in some public way.
6. 2poly-to-2sum is run. If there are misbehaving servers, their indices are removed from $\mathscr{G}$ and the protocol loops to Step 5.
7. 2sum-to-2poly is run. If there are misbehaving servers (among Λ), their indices are removed from $\mathscr{G}$ and the protocol loops to Step 5.
8. All previous share information is erased.

Fig. 10. DL-based proactive maintenance (key update) protocol.

1. *Initial configuration*: DL-based system parameters: (p,q,g).
2. The dealer generates $h\in_R Z_p^*$, $x,x'\in_R Z_q$, $y=g^x$, and a (t,l)-US-VSS on secrets x,x'.
3. Each (ordered) pair of servers (S_i,S_j) performs ZKSETUP-DL_{S_i,S_j} (p,q,g,h).
4. Each server maintains a list $\mathscr{G}$ of server indices for servers that have not misbehaved (i.e., they are considered good). It also maintains public parameters: (p,q,g,h,y,l,t) and the verification shares of the (t,l)-US-VSS polynomials $a(\ )$ and $a'(\ )$ which have $a(0)=x$ and $a'(0)=x'$.
5. When a message m needs to be signed, the servers agree on the public parameters, then the function application protocol is run.
6. When an update is scheduled to occur, the servers agree on the public parameters, then the proactive maintenance protocol is run.

Fig. 11. DL-based proactive threshold protocol.

5. Proofs

We start this section by proving certain properties of the basic techniques we use, and then we prove the main theorems stated in the introduction.

5.1. Proofs of techniques

The following lemma from Frankel et al. [20] is used to prove the subsequent lemma, which in turn is used to prove the simulatability of INT-(t,l)-US-VSS.

Lemma 1. *Let* $r(x)=r_0+r_1x+\cdots+r_tx^t$ *be a random polynomial of degree* t *such that* $r(0)=r_0=L^2k$ $(k\in[0,K])$ *and* $r_j\in_R\{0,L,\ldots,\beta L^3K\}$ *for* $1\leqslant j\leqslant t$. *Let* Λ' *be a*

set of t servers. Then with probability at least $1-2t/\beta$, *for any* $\hat{k}\in[0,K]$, *there exists a polynomial* $r'(x)=r'_0+r'_1x+\cdots+r'_tx^t$ *with* $r'(0)=r'_0=L^2\hat{k}$ *and* $r'_j\in\{0,L,\ldots,\beta L^3K\}$ *for* $1\leqslant j\leqslant t$ *such that* $r(i)=r'(i)$ *for* $i\in\Lambda'$.

Lemma 2. *Let* $\gamma\in[1,m]$. *Let* $r(x)=r_0+r_1x+\cdots+r_tx^t$ *be a random polynomial of degree t such that* $r(0)=r_0=L^2k$ $(k\in[0,K])$ *and* $r_j\in_R\{0,L,\ldots,\beta L^3K\}$ *for* $1\leqslant j\leqslant t$. *Let* $r'(x)=r'_0+r'_1x+\cdots+r'_tx^t$ *be a random polynomial of degree t such that* $r'(0)=r'_0=L^2k'$ $(k'\in[0,mK\beta])$ *and* $r_j\in_R\{0,L,\ldots,\beta L^3mK\}$ *for* $1\leqslant j\leqslant t$. *Let* Λ' *be a set of t servers. Then with probability at least* $1-(4t+2)/\beta$, *for any* $\hat{k}\in[0,K]$, *there exists polynomials* $\hat{r}(x)=\hat{r}_0+\hat{r}_1x+\cdots+\hat{r}_tx^t$ *with* $\hat{r}(0)=\hat{r}_0=L^2\hat{k}$ *and* $\hat{r}_j\in\{0,L,\ldots,\beta L^3K\}$ *for* $1\leqslant j\leqslant t$, *and* $\hat{r}'(x)=\hat{r}'_0+\hat{r}'_1x+\cdots+\hat{r}'_tx^t$ *with* $\hat{r}'(0)=L^2(\gamma k+k'-\gamma\hat{k})$, $0\leqslant\hat{r}'(0)\leqslant\beta L^2mK$ *and* $\hat{r}'_j\in\{0,L,\ldots,\beta L^3mK\}$ *for* $1\leqslant j\leqslant t$, *such that* $r(i)=\hat{r}(i)$ *and* $r'(i)=\hat{r}'(i)$ *for* $i\in\Lambda'$, *and* $\gamma r(x)+r'(x)=\gamma\hat{r}(x)+\hat{r}'(x)$.

Proof. Except for the last equation, we get from Lemma 1 that the probability that the polynomials $\hat{r}(x)$ and $\hat{r}'(x)$ (with coefficients in the correct ranges) do not exist is at most $[2/\beta]+[2t/\beta]+[2t/\beta]$, where the first $2/\beta$ arises from the probability that $\hat{r}'(0)$ is in the correct range, given an additive offset of $L^2(\gamma k-\gamma\hat{k})\in[-L^2mK,L^2mK]$ from L^2k'. If those polynomials do exist, then the last equation follows since (1) the degree of the polynomial on each side of the equivalence is t, (2) the polynomials obviously agree at the t locations in Λ', and (3) the polynomials agree at 0, since $\gamma r(0)+r'(0)=L^2\gamma k+L^2k'=L^2\gamma\hat{k}+L^2(\gamma k+k'-\gamma\hat{k})=\gamma\hat{r}(0)+\hat{r}'(0)$. □

5.2. Useful RSA lemmas

Lemma 3 (Frankel et al. [24]). *Let k be the security parameter. Let modulus generator GE define a family of modulus generating functions* (*i.e.*, $N\leftarrow GE(1^k)$ *be an RSA modulus with security parameter k*). *For any probabilistic polynomial-time adversary* $\mathscr{A}$, *the following is negligible*: $\Pr[u^e\equiv w^d \bmod N;(e\neq 0)\vee(d\neq 0):N\leftarrow GE(1^k);u, w\in_R\{0,1\}^k;e,d\leftarrow A(1^k,w,u,N)]$

Proof. Similar to Bach [2]. □

The following corollary follows from Lemma 3 and the RSA assumption (and hence from the RSA assumption).

Corollary 1. *Let k be the security parameter. Let GE be an RSA generator* (*that produces large public exponents*), *i.e.*, $(N,e)\leftarrow GE(1^k)$. *For any probabilistic polynomial-time algorithm A*,

$$\Pr[(g^\alpha\equiv h^\beta \bmod N;(\alpha\neq 0)\vee(\beta\neq 0))\vee(g=u^e):\\ (N,e)\leftarrow GE(1^k);g,h\in_R\{0,1\}^k;(\alpha,\beta,u)\leftarrow A(1^k,g,h,N)]$$

is negligible.

5.3. Proofs of protocols

For clarity our proofs are written for distributed signature schemes. It should be easy to see that with appropriate wording changes ("ciphertext" for "message", etc.) that they also apply to distributed decryption schemes.

Proof of Theorem 1. We prove the robustness and security of the basic DL-based proactive threshold protocol. Robustness is based on the DLP Assumption, while security is based on the security of the underlying signature scheme. Recall that we assume the adversary is mobile and adaptive.

Robustness. Say $P(k)$ is a polynomial bound on the number of messages m that the protocol signs. Say an adversary prevents the signing of a message with non-negligible probability ρ. We will show how to solve the DLP with probability

$$\rho - \frac{tP(k)+1}{q}.$$

Say we are given an instance of the DLP, namely for a given set of parameters (p,q,g), we are given a uniformly chosen value $h \in G_q$. We create a simulator that runs the dealer as normal, except that h is taken from the DLP instance (when h is generated in a distributed way, say as in Gennaro et al. [29] or Frankel et al. [23], we would also be able to inject our DLP instance). Then the servers are run as normal, except that the extractor for the ZKPROOF-DL-REP protocol is run whenever an uncorrupted server is playing the verifier and a corrupted server is playing the prover. Note that since the simulator knows the secrets x, x', the normal operation of the servers can be simulated easily. We will show that if an adversary is able to prevent a message from being signed, we can (except with negligible probability) determine $\mathrm{DL}(h,g)$.

If a server is not corrupted, the probability of a failed extraction using that server is $1/q$. There will obviously be at least one uncorrupted server that runs the extractor with every corrupted server, and thus the probability of any corrupted server not allowing a successful extraction during the protocol is at most $tP(k)/q$. Say S_i runs the extractor successfully on S_j. If S_i extracts a way to open the commitment $C_{i,j} = g^{\sigma_{i,j}} h^{\sigma'_{i,j}}$ (from the setup protocol), say with $(\tau_{i,j}, \tau'_{i,j})$, then except with probability $1/q$, this will give

$$\mathrm{DL}(h,g) = \frac{\tau_{i,j} - \sigma_{i,j}}{\sigma'_{i,j} - \tau'_{i,j}}.$$

Therefore, with probability at most

$$\frac{tP(k)+1}{q},$$

there was either an extraction that failed or an extraction that succeeded, but produced a way to open $C_{i,j}$ with the same pair of values used to create it.

It should be clear that in any update period, after at most $t+1$ attempts, there will be a set Λ of $t+1$ servers that participate in the update without any verification failures. Now say that the polynomial shares created during some operational period

do not interpolate to $v(0)=x$ and $v'(0)=x'$. (Note that the verification share for the zero coefficients of the polynomials in every operational period will equal $g^x h^{x'}$.) Let $\{(\hat{d}_j,\hat{d}'_j)\}_{j\in\Lambda}$ be the interpolated *dual shares* of $\{v_j(\ ),v'_j(\ )\}_{j\in\Lambda}$ (which can be found from the shares of $t+1$ good servers). It is easy to see that

$$\begin{aligned} g^x h^{x'} &= g^{v(0)} h^{v'(0)} \\ &= \prod_{j\in\Lambda} g^{v_j(0)} h^{v'_j(0)} \\ &= \prod_{j\in\Lambda} g^{\hat{d}_j} h^{\hat{d}'_j}. \end{aligned}$$

We also have

$$\sum_{j\in\Lambda} \hat{d}_j \neq x$$

and

$$\sum_{j\in\Lambda} \hat{d}'_j \neq x'.$$

But then

$$DL(h,g) = \frac{(\sum_{j\in\Lambda} \hat{d}_j) - x}{x' - \sum_{j\in\Lambda} \hat{d}'_j} \bmod q.$$

Now say the adversary prevents a message m from being signed. It should be clear that after at most $t+1$ attempts, there will be a set Λ of $t+1$ servers that participate in signing m without any verification failures. This implies that the signature obtained must be incorrect. Let $\{E_i\}_{i\in\Lambda}$ be the check shares for the (single) additive shares in this signing attempt. If $\prod_{i\in\Lambda} E_i = y$, then the signature must be correct, so we may assume $\prod_{i\in\Lambda} E_i \neq y$. Let $\{(\delta_i,\delta'_i)\}_{i\in\Lambda}$ be the extracted (or simulator generated, for uncorrupted servers) dual-shares. It is easy to see that

$$g^x h^{x'} = \prod_{i\in\Lambda} B_i = \prod_{i\in\Lambda} D_i = \prod_{i\in\Lambda} g^{\delta_i} h^{\delta'_i}.$$

We also have $g^x = y \neq \prod_{i\in\Lambda} E_i = \prod_{i\in\Lambda} g^{\delta_i}$. But then

$$g^x h^{x'} = g^{\sum_{i\in\Lambda}\ \delta_i} h^{\sum_{i\in\Lambda}\ \delta'_i},$$

with $x \neq \sum_{i\in\Lambda} \delta_i$ (and hence $x' \neq \sum_{i\in\Lambda} \delta'_i$), and thus

$$\mathrm{DL}\,(h,g) = \frac{(\sum_{i\in\Lambda}\ \delta_i) - x}{x' - \sum_{i\in\Lambda}\ \delta'_i} \bmod q.$$

Therefore, with probability $\rho - (tP(k)+1)/q$, $\mathrm{DL}(h,g)$ can be found.

Security. Here we show that if the adversary can sign a new message, then it can break the security of the (non-distributed) signature scheme w.r.t. the same attack [32].

Say an adversary can sign a new message in the DL-based proactive threshold protocol with non-negligible probability ρ. We will show that we can sign a new message in the underlying signature scheme with probability ρ. Say the signature scheme has parameters (p,q,g,y). Then we create the following simulator:

1. *Initialization*: The signature scheme gives parameters (p,q,g,y)
2. Simulate the dealer by generating $h \in_R Z_p^*$, $x' \in_R Z_q$, and producing a (t,l)-US-VSS with polynomials $(\hat{a}(\ ),\hat{a}'(\ ))$ on secrets $0,x'$ (i.e., $\hat{a}(0)=0$ and $\hat{a}'(0)=x'$).
3. Each (ordered) pair of servers performs the ZKSETUP-DL protocol, using g and h as the generators, except that an uncorrupted verifier interacting with a corrupted prover uses the extractor to determine how to open the commitment for that prover.
4. Each server maintains a list $\mathcal{G}$ of server indices for servers that have not misbehaved (i.e., they are considered good).
5. When a message m needs to be signed, the following DISTAPPLY protocol is run:
   (a) A set $\Lambda \subseteq G$ with $|\Lambda| = t+1$ is chosen in some public way.
   (b) 2poly-to-2sum is performed using the simulator-generated (t,l)-US-VSS with polynomials $(\hat{a}(\ ),\hat{a}'(\ ))$, producing values $(\hat{b}_j,\hat{b}'_j)$ for $j \in \Lambda$, along with their associated check shares. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 5a.
   (c) An index of a faking server, say i, is picked at random from Λ. 2sum-to-2sum is performed using the simulator-generated values $\{(\hat{b}_j,\hat{b}'_j)\}_{j\in\Lambda}$ and their associated check shares from the previous step, producing values $(\hat{d}_j,\hat{d}'_j)$ for $j \in \Lambda$, along with their associated check shares. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 5a. If S_i is compromised, then the simulation rewinds to Step 5c, and is attempted again.
   (d) For each $j \in \Lambda\backslash\{i\}$, S_j performs 2sum-to-1sum using the simulator-generated values $(\hat{d}_j,\hat{d}'_j)$ and their associated check shares from the previous step. S_i, however, produces $\hat{E}_i = yg^{\hat{d}_j}$ and in each ZKPROOF-DL-REP actually proves knowledge of how to open the commitment, instead of knowledge of the discrete log of $\hat{E}_i$. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 5a. If S_i is compromised, then the simulation rewinds to Step 5c, and is attempted again.
   (e) The additive application step of the signature protocol is run, but with S_i simulated by using the signature on m obtained from the signature oracle. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 5a. If S_i is compromised, then the simulation rewinds to Step 5c, and is attempted again.
   (f) So as not to bias the simulation with respect to the number of corruptions of the adversary during the DISTAPPLY protocol, we randomly choose an index of one of the remaining non-corrupted servers from Λ, and if it is not i, the simulation rewinds to Step 5c and is attempted again.

6. For each update period, the following DISTUPDATE protocol is run:
   (a) Each (ordered) pair of servers performs the ZKSETUP-DL protocol, except that, as before, an uncorrupted verifier interacting with a corrupted prover uses the extractor to determine how to open the commitment for that prover.
   (b) A set $\Lambda \subseteq \mathscr{G}$ with $|\Lambda| = t+1$ is chosen in some public way.
   (c) 2poly-to-2sum is performed using the simulator-generated (t,l)-US-VSS with polynomials $(\hat{a}(\ ), \hat{a}'(\ ))$, producing values $(\hat{b}_j, \hat{b}'_j)$ for $j \in \Lambda$, along with their associated check shares. If there are misbehaving servers, their indices are removed from $\mathscr{G}$ and the protocol loops to Step 6b.
   (d) 2sum-to-2poly is performed using the simulator-generated values $(\hat{b}_j, \hat{b}'_j)$ for $j \in \Lambda$ and their associated check shares. (Note that we use call these "b" values instead of "d" values, as they are called in the 2sum-to-2poly protocol.) If there are misbehaving servers, their indices are removed from $\mathscr{G}$ and the protocol loops to Step 6b.

Note that the probability of rewinding is exactly $t/(t+1)$, and thus the simulator requires on average a factor of at most $t+1$ more time than the real protocol. Thus the simulation is polynomial time. The simulation is perfect if all the extractors succeed. Since the extractors have to be run at most ltU times (where U is the number of update periods), and the probability of a single extractor failing is $1/q$, the probability of distinguishing a simulated view from a real view is ltU/q. Thus the simulation is statistically indistinguishable from the real protocol.

If the adversary is able to then generate a new signature, then it is clear that we would have an algorithm to break the signature scheme. □

Proof of Theorem 2. We prove the robustness and security of the basic RSA-based threshold protocol. Security is by direct reduction from RSA, while robustness is based on Corollary 1. (We will call this the *Corollary* 1 *assumption*.) Recall that we assume the adversary is stationary and adaptive. We will assume that the public key e is large ($\Theta(k)$ bits). (For small e we can use a technique from Cramer et al. [11] to obtain ZK proofs that allow us to prove similar results.)

Security. Here we reduce the security of RSA to the security of our basic RSA-based threshold protocol. Say an adversary, after watching polynomially many messages be signed in the basic RSA-based threshold protocol, can sign a new challenge message. with non-negligible probability ρ. Then we will give a polynomial-time algorithm to break RSA with probability close to ρ.

Say we are given an RSA key (N,e) and a challenge message m^* to be signed. We will run the adversary against a simulation of the protocol, and then present m^* to be signed. We will show that the probability that an adversary can distinguish the simulation from the real protocol is negligible, and thus the probability that it signs m^* is negligibly less than ρ.

The simulator is as follows:

1. *Initialization*: The RSA parameters (N,e) are given. We may also assume that we have a list of random message signature pairs $\{(m, m^{1/e})\}$.

2. Simulate the dealer by computing the public value x^* (using public values N, e, L, as in the real protocol) generating $g, h \in_R Z_N^*$, $x' \in_R Z_{N^3}$, and producing a INT-(t, l)-US-VSS with polynomials $(\hat{a}(\ ), \hat{a}'(\ ))$ on secrets $0, x'$ (i.e., $\hat{a}(0) = 0$ and $\hat{a}'(0) = x'$).
3. Each (ordered) pair of servers performs the ZKSETUP-RSA protocol, using g and h as the generators, except that an uncorrupted verifier interacting with a corrupted prover uses the extractor to determine how to open the commitment for that prover.
4. Each server maintains a list $\mathcal{G}$ of server indices for servers that have not misbehaved (i.e., they are considered good).
5. When a message m needs to be signed, the following DISTAPPLY protocol is run:
   (a) A set $\Lambda \subseteq \mathcal{G}$ with $|\Lambda| = t + 1$ is chosen in some public way.
   (b) 2poly-to-2sum is performed using the simulator-generated INT-(t, l)-US-VSS with polynomials $(\hat{a}(\ ), \hat{a}'(\ ))$, producing values $(\hat{b}_j, \hat{b}'_j)$ for $j \in \Lambda$, along with their associated check shares. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 5a.
   (c) An index of a faking server, say i, is picked at random from Λ. 2sum-to-2sum is performed using the simulator-generated values $\{(\hat{b}_j, \hat{b}'_j)\}_{j \in \Lambda}$ and their associated check shares from the previous step, producing values $(\hat{d}_j, \hat{d}'_j)$ for $j \in \Lambda$, along with their associated check shares. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 5a. If S_i is compromised, then the simulation rewinds to Step 5c, and is attempted again.
   (d) For each $j \in \Lambda \backslash \{i\}$, S_j performs 2sum-to-1sum using the simulator-generated values $(\hat{d}_j, \hat{d}'_j)$ and their associated check shares from the previous step. S_i, however, produces $\hat{E}_i = m^{(1/e)-x^*} m^{\hat{d}_j}$ and in each ZKPROOF-IF-REP actually proves knowledge of how to open the commitment, instead of knowledge of the discrete log of $\hat{E}_i$. If there are misbehaving servers, their indices are removed from $\mathcal{G}$ and the protocol loops to Step 5a. If S_i is compromised, then the simulation rewinds to Step 5c, and is attempted again.
   (e) So as not to bias the simulation with respect to the number of corruptions of the adversary during the DISTAPPLY protocol, we randomly choose an index of one of the remaining non-corrupted servers, and if it is not i, the simulation rewinds to Step 5c and is attempted again.

Note that the probability of rewinding is exactly $t/(t+1)$, and thus the simulator requires on average a factor of at most $t + 1$ more time than the protocol. Thus the simulation is polynomial time. The probability of distinguishing the simulation from the real protocol is at most the probability of distinguishing the simulated INT-(t, l)-US-VSS from the real one, plus the probability of an extractor failing. All of this can be bounded by $(4t + 2)/N + tl/e$. Thus with probability negligibly less than ρ, we can generate a signature on m^*.

Robustness. Say $P(k)$ is a polynomial bound on the number of messages m that the protocol signs. Say an adversary prevents the signing of a message with non-negligible

probability ρ. We will show how to break the Corollary 1 assumption with probability

$$\rho - \frac{2tP(k)+1}{e}.$$

Say an RSA public key was generated $(N,e) \leftarrow GE(1^k)$ and we are given a uniformly chosen $g,h \in Z_N^*$, as in Corollary 1. We use the simulator that is used to prove security, except that g,h are taken from the RSA instance, and the extractor for the ZKPROOF-IF-REP protocol is run whenever an uncorrupted server is playing the verifier and a corrupted server is playing the prover. We will show that if an adversary is able to prevent a message from being signed, we can (except with negligible probability) either find α,β such that $g^\alpha \equiv h^\beta \bmod N$ or find u such that $u^e \equiv g \bmod N$.

If a server is not corrupted, the probability of a failed extraction using that server is $1/e$. There will obviously be at least one uncorrupted server that runs the extractor with every corrupted server, and thus the probability of any corrupted server not allowing a successful extraction during the protocol is at most $tP(k)/e$. Say S_i runs the extractor successfully on S_j. If S_i extracts a way to open the commitment $C_{i,j} = g^{\sigma_{i,j}}(\sigma'_{i,j})^e$, say with $(\tau_{i,j},\tau'_{i,j})$, then this will give

$$g^{\sigma_{i,j}}(\sigma'_{i,j})^e = g^{\tau_{i,j}}(\tau'_{i,j})^e,$$

and thus

$$g^{\sigma_{i,j}-\tau_{i,j}} = (\tau'_{i,j}/\sigma'_{i,j})^e.$$

Except with probability $1/e$, $\gcd(\sigma_{i,j} - \tau_{i,j}, e) = 1$, so using the extended Euclidean algorithm, one can compute α,β such that $\alpha e + 1 = \beta(\sigma_{i,j} - \tau_{i,j})$, and thus

$$g^{1/e} = \frac{(\tau'_{i,j}/\sigma'_{i,j})^\beta}{g^\alpha}.$$

Therefore, assuming the adversary has not distinguished the simulation from the real protocol, with probability at most

$$\frac{tP(k)+1}{e},$$

there was either an extraction that failed or an extraction that succeeded, but produced a way to open $C_{i,j}$ with an RSA–REP pair that did not allow one to compute the RSA inverse of g. Recall that the probability the adversary can distinguish the simulation from the real protocol is at most

$$\frac{4t+2}{N} + \frac{tl}{e}.$$

Now say the adversary prevents a message m from being signed. It should be clear that after at most $t+1$ attempts, there will be a set Λ of $t+1$ servers that participate in signing m without any verification failures. This implies that the signature obtained must be incorrect. Let $\{E_j\}_{j\in\Lambda}$ be the check shares for the (single) additive shares in this signing attempt for message m. (Recall that for the faking server

S_i, $E_i = m^{(1/e)-x^*} m^{d_i} \bmod N$.) Then $m^{x^*} \prod_{j\in\Lambda} E_j \neq m^{1/e}$. Let $\{(\delta_j, \delta'_j, \Delta_j)\}_{j\in\Lambda}$ be the extracted (or previously known to the simulator, for uncorrupted servers) *dual shares,* along with the exponent on D_j. (For faking server S_i, $\delta_i = d_i$, $\delta'_i = d'_i$ and $\Delta_i = 1$.) Let $\Delta = \prod_{j\in\Lambda} \Delta_j$. It is easy to see that

$$(h^{x'})^{\Delta} = \left(\prod_{j\in\Lambda} B_j\right)^{\Delta} = \left(\prod_{j\in\Lambda} D_j\right)^{\Delta} = \prod_{j\in\Lambda} (g^{\delta_j} h^{\delta'_j})^{\Delta/\Delta_j}.$$

We also have

$$(m^{1/e})^{\Delta} \neq \left(m^{x^*} \prod_{j\in\Lambda} E_j\right)^{\Delta} = (m^{x^*})^{\Delta} (m^{(1/e)-x^*} m^{\delta_i})^{\Delta} \prod_{j\in\Lambda\setminus\{i\}} (m^{\delta_j})^{\Delta/\Delta_j}.$$

But then

$$h^{x'\Delta} = g^{\sum_{j\in\Lambda}(\delta_j \Delta/\Delta_j)} h^{\sum_{j\in\Lambda}(\delta'_j \Delta/\Delta_j)},$$

with $0 \neq \sum_{j\in\Lambda}(\delta_j \Delta/\Delta_j)$ (and hence $x'\Delta \neq \sum_{j\in\Lambda}(\delta'_j \Delta/\Delta_j)$), and thus

$$g^{\sum_{j\in\Lambda} \delta_j \Delta/\Delta_j} = h^{x'\Delta - \sum_{j\in\Lambda} \delta'_j \Delta/\Delta_j}.$$

Thus, assuming the adversary can prevent a message from being signed, the probability of finding either α, β such that $g^{\alpha} \equiv h^{\beta} \bmod N$ or u such that $u^e \equiv g \bmod N$ is at least the non-negligible probability

$$\rho - \frac{tP(k) + 1 + tl}{e} + \frac{4t + 2}{N},$$

contradicting Corollary 1. □

This concludes the proofs showing that the above protocols are robust and secure.

6. Conclusion

To summarize, we have provided protocols for distributed public-key systems that are adaptively secure. Our techniques and protocols are efficient and typically take constant communication rounds when there are no faults (and a fault may cause a constant delay).

Appendix A. ZK proofs

We use efficient ZK proofs of knowledge (POKs) derived from Frankel et al. [24] and Cramer et al. [11]. These are composed of combinations of Σ-protocols [10] (i.e., Schnorr-type proofs [41]). For each ZK proof that we need, we will have a separate *proof* protocol, but there will be a single *setup* protocol used for all ZK proofs. Say A wishes to prove knowledge of X to B. Then the setup protocol will consist of B making a commitment and proving that he can open it in a witness indistinguishable

way [19], and the proof protocol will consist of A proving to B either the knowledge of X or that A can open the commitment. (See [11] for details) This construction allows the proof protocols to be run concurrently without any timing constraints, as long as they are run after all the setup protocols have completed. (For more on the problems encountered with concurrent ZK proofs see [36, 16, 17].)

The DL-based and RSA-based ZK-proof-setup protocols are exactly the Σ-protocols for commitments over q-one-way-group-homomorphisms (q-OWGH), given in [11]. Recall the q-OWGH for a DL-based system with parameters (p,q,g) is $f(x)=g^x \bmod p$, and the q-OWGH for an RSA-based system with parameters (N,e) is $f(x)=x^e \bmod N$ (with $q=e$ in this case).

Let KE denote the *knowledge error* of a POK.

Formally, we define ZKSETUP-DL$_{A,B}(p,q,g,h)$ as a protocol in which A generates a commitment C and engages B in a witness-hiding (WH) POK ($KE=1/q$) of $\sigma,\sigma' \in Z_q$ where $C \equiv g^{\sigma}h^{\sigma'} \bmod p$.

We define ZKSETUP-RSA$_{A,B}(N,e,g)$ as a protocol in which A generates a commitment C and engages B in a WH POK ($KE=1/e$)[10] of (σ,σ') (with $\sigma \in Z_e$, $\sigma' \in Z_N^*$) where $C \equiv g^{\sigma}(\sigma')^e \bmod N$.

We define ZKPROOF-DL$_{A,B}(p,q,g,h,D)$ as a protocol in which A engages B in a WH POK ($KE=1/q$) of either $d \in Z_q$ where $D \equiv g^d \bmod p$, or $\tau,\tau' \in Z_q$ where $C_{B,A} \equiv g^{\tau}h^{\tau'} \bmod p$ and $C_{B,A}$ is the commitment generated in ZKSETUP-DL$_{B,A}(p,q,g,h)$.

We define ZKPROOF-DL-REP$_{A,B}(p,q,g,h,E,h',D)$ as a protocol in which A engages B in a WH POK ($KE=1/q$) of either $d,d' \in Z_q$ where $D \equiv g^d(h')^{d'} \bmod p$ and $E \equiv g^d \bmod p$, or $\tau,\tau' \in Z_q$ where $C_{B,A} \equiv g^{\tau}h^{\tau'} \bmod p$ and $C_{B,A}$ is the commitment generated in ZKSETUP-DL$_{B,A}(p,q,g,h)$.

We define[11] ZKPROOF-IF-REP$_{A,B}(N,e,g,m,E,h,D)$ as a protocol in which A, who knows integers $d \in (-a,a]$ and $d' \in (-b,b]$ such that $E \equiv m^d \bmod N$ and $D \equiv g^d h^{d'} \bmod N$, engages B in a WH POK ($KE=1/e$) of either $\Delta \in Z_e$, $\delta \in (-2ae(N+1), 2ae(N+1)]$, and $\delta' \in (-2be(N+1), 2be(N+1)]$ where $D^{\Delta} \equiv g^{\delta}h^{\delta'} \bmod N$ and $E^{\Delta} \equiv g^{\delta} \bmod N$, or (τ,τ') (with $\tau \in Z_e$, $\tau' \in Z_N^*$) where $C_{B,A} \equiv g^{\tau}(\tau')^e \bmod N$ and $C_{B,A}$ is the commitment generated in ZKSETUP-RSA$_{B,A}(N,e,g)$. This protocol is honest-verifier statistical zero-knowledge with a statistical difference between the distribution of views produced by the simulator and in the real protocol bounded by $2/N$.

A.1. Proof of representations

Here we give the main Σ-protocol used in ZKPROOF-IF-REP$_{A,B}(N,e,g,m,E,h,D)$.[12]

1. Initially, the parameters (N,e,g,m,E,h,D) are public, and A knows integers $d \in (-a,a]$ and $d' \in (-b,b]$ such that $E \equiv m^d \bmod N$ and $D \equiv g^d h^{d'} \bmod N$.

[10] This implies e must be exponentially large in the security parameter k in order to obtain a sound proof. However, if e is small (say $e=3$) we can use different setup and proof protocols described in [11] to obtain provably secure and robust RSA-based protocols.

[11] IF stands for *integer factorization*.

[12] Recall that this main protocol is combined with a Σ-protocol proving knowledge of a commitment generated in a setup protocol, using an *OR* construction.

2. A generates $r \in_R (-aeN, aeN]$ and $r' \in (-beN, beN]$, computes $V = m^r \bmod N$ and $W = g^r h^{r'} \bmod N$, and sends V, W to B.
3. B generates $c \in_R Z_e$ and sends c to A.
4. A computes $z = cd + r$ and $z' = cd' + r'$, and sends z, z' to B.
5. B checks that $m^z \equiv E^c V \bmod N$ and $g^z h^{z'} = D^c W \bmod N$.

In all steps, A and B also check that the values received are in the appropriate ranges.

The above is a POK of $\Delta \in Z_e$, $\delta \in (-2ae(N+1), 2ae(N+1)]$, and $\delta' \in (-2be(N+1), 2be(N+1)]$ in which $m^\delta \equiv E^\Delta \bmod N$ and $g^\delta h^{\delta'} = D^\Delta \bmod N$. The knowledge error is $1/e$, and the protocol is honest-verifier statistical zero-knowledge, with a statistical difference between views produced by the simulator and those in the real protocol bounded by $2/N$.

References

[1] G. Agnew, R.C. Mullin, S. Vanstone, Improved digital signature scheme based on discrete exponentiation, Electron. Lett. 26 (1990) 1024–1025.

[2] E. Bach, Discrete logarithms and factoring, Technical report, Computer Science Division (EECS), University of California Press, Berkeley, CA, June 1984.

[3] D. Beaver, Adaptively secure oblivious transfer, in: Advances in Cryptology—ASIACRYPT '98, Lecture Notes in Computer Science, Springer, Berlin, November 1998, pp. 300–314.

[4] D. Beaver, S. Haber, Cryptographic protocols provably secure against dynamic adversaries, in: Advances in Cryptology—EUROCRYPT 92, Lecture Notes in Computer Science, Vol. 658, Springer, Berlin, 24–28 May 1992, pp. 307–323.

[5] R. Canetti, U. Feige, O. Goldreich, M. Naor, Adaptively secure multi-party computation, in: STOC'96, Proc. 28th Annu. ACM Symp. on the Theory of Computing, Philadelphia, PA, 22–24 May 1996, pp. 639–648.

[6] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Adaptive security of threshold systems, in: M. Wiener (Ed.), Advances in Cryptology—CRYPTO '99, Lecture Notes in Computer Science, Vol. 1666, Springer, Berlin, 15–19 August 1999, pp. 98–115.

[7] R. Canetti, S. Goldwasser, An efficient threshold public key cryptosystem secure against adaptive chosen message attack, Advances in Cryptology—EUROCRYPT 99, Lecture Notes in Computer Science, Vol. 1592, Springer, Berlin, 1999, pp. 90–106.

[8] R. Canetti, S. Halevi, A. Herzberg, Maintaining authenticated communication in the presence of break-ins, PODC'97, Proc. 16th Annu. ACM Symp. on Principles of Distributed Computing, 1997, pp. 15–24.

[9] J.D. Cohen, M.J. Fischer, A robust and verifiable cryptographically secure election scheme (extended abstract), in: 26th Annu. Symp. on Foundations of Computer Science, Portland, OR, 21–23 October 1985, IEEE Press, New York, pp. 372–382.

[10] R. Cramer, Modular Design of Secure Yet Practical Cryptographic Protocols, Ph.D. thesis, University of Amsterdam, 1995.

[11] R. Cramer, I. Damgård, P. MacKenzie, Efficient zero-knowledge proofs of knowledge without intractability assumptions, in: H. Imai, Y. Zheng (Eds.), Advances in Public Key Cryptography—PKC 2000, Lecture Notes in Computer Science, Vol. 1751, Springer, Berlin, January 2000, pp. 354–372.

[12] A. De Santis, Y. Desmedt, Y. Frankel, M. Yung, How to share a function securely (extended summary), in: Proc. 26th Annu. ACM Symp. on the Theory of Computing, Montréal, Qué., Canada, 23–25 May 1994, pp. 522–533.

[13] Y. Desmedt, Y. Frankel, Threshold cryptosystems, in: CRYPTO'89 Advances in Cryptology—CRYPTO '89, Lecture Notes in Computer Science, Vol. 435, 20–24 August 1989, Springer, Berlin, 1990, pp. 307–315.

[14] Y. Desmedt, Y. Frankel, Shared generation of authenticators and signatures (extended abstract), in: CRYPTO'91, Advances in Cryptology—CRYPTO '91, Lecture Notes in Computer Science, Vol. 576, 11–15 August 1991, Springer, Berlin, 1992, pp. 457–469.

[15] W. Diffie, M. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory 22 (6) (1976) 644–654.
[16] C. Dwork, M. Naor, A. Sahai, Concurrent zero-knowledge, in: STOC'98, Proc. 30th Annu. ACM Symp. on Theory of Computing, Dallas, TX, 23–26 May 1998, pp. 409–418.
[17] C. Dwork, A. Sahai, Concurrent zero-knowledge: reducing the need for timing constraints, in: H. Krawczyk (Ed.), Advances in Cryptology—CRYPTO '98, Lecture Notes in Computer Science, Vol. 1462, Springer, Berlin, 17–21 August 1998, pp. 442–457.
[18] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithm, IEEE Trans. Inform. Theory 31 (1985) 469–472.
[19] U. Feige, A. Shamir, Zero knowledge proofs of knowledge in two rounds, in: CRYPTO'89, Advances in Cryptology—CRYPTO '89, Lecture Notes in Computer Science, Vol. 435, 20–24 August 1989, Springer, Berlin, 1990, pp. 526–545.
[20] Y. Frankel, P. Gemmell, P.D. MacKenzie, M. Yung, Optimal-resilience proactive public-key cryptosystems, in: 38th Annu. Symp. on Foundations of Computer Science, Miami Beach, FL, IEEE Press, New York, 20–22 October 1997, pp. 384–393.
[21] Y. Frankel, P. Gemmell, P.D. MacKenzie, M. Yung, Proactive RSA, in: Advances in Cryptology—CRYPTO '97, Lecture Notes in Computer Science, Vol. 1294, Springer, Berlin, 17–21 August 1997, pp. 440–454.
[22] Y. Frankel, P. Gemmell, M. Yung, Witness-based cryptographic program checking and robust function sharing, in: STOC'96, Proc. 28th Annu. ACM Symp. on the Theory of Computing, Philadelphia, PA, 22–24 May 1996, pp. 499–508.
[23] Y. Frankel, P.D. MacKenzie, M. Yung, Adaptively-secure distributed public-key systems, in: European Symposium on Algorithms—ESA '99, Lecture Notes in Computer Science, Vol. 1643, Springer, Berlin, 16–18 July 1998, pp. 4–27.
[24] Y. Frankel, P.D. MacKenzie, M. Yung, Robust efficient distributed RSA-key generation, in: STOC'98, Proc. 30th Annu. ACM Symp. on Theory of Computing, Dallas, TX, 23–26 May 1998, pp. 663–672.
[25] Y. Frankel, P.D. MacKenzie, M. Yung, Adaptively-secure optimal-resilience proactive RSA, in: Advances in Cryptology—ASIACRYPT '99, Lecture Notes in Computer Science, Springer, Berlin, November 1999.
[26] Y. Frankel, M. Yung, Distributed public-key cryptosystems, in: H. Imai, Y. Zheng (Eds.), Advances in Public Key Cryptography—PKC '98, Lecture Notes in Computer Science, Vol. 1431, Springer, Berlin, Febuary 1998, pp. 1–13 (invited talk).
[27] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Robust and efficient sharing of RSA functions, in: Advances in Cryptology—CRYPTO '96, Notes in Computer Science, Vol. 1109, Springer, Berlin, 18–22 August 1996, pp. 157–172.
[28] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Robust threshold DSS signatures, in: Advances in Cryptology—EUROCRYPT 96, Lecture Notes in Computer Science, Vol. 1070, Springer, Berlin, 12–16 May 1996, pp. 354–371.
[29] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, The (in)security of distributed key generation in dlog-based cryptosystems, in: Advances in Cryptology—EUROCRYPT 99, Lecture Notes in Computer Science, Vol. 1592, Springer, Berlin, 1999, pp. 295–310.
[30] S. Goldwasser, Multi-party computations: past and present, in: PODC'97, Proc. 16th Annu. ACM Symp. on Principles of Distributed Computing, 1997, pp. 1–6 (invited talk).
[31] L. Harn, Group oriented (t,n) digital signature scheme, IEEE Proc. Comput. Digit. Tech. 141 (5) (1994) 307–313.
[32] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, M. Yung, Proactive public-key and signature schemes, in: Proc. 3rd Annu. ACM Conf. on Computer and Communications Security, 1996, pp. 100–110.
[33] A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung, Proactive secret sharing, or: How to cope with perpetual leakage, in: Advances in Cryptology—CRYPTO '95, Lecture Notes in Computer Science, Vol. 963, Springer, Berlin, 27–31 August 1995, pp. 339–352.
[34] S. Jarecki, Proactive secret sharing and public key cryptosystems, Master's thesis, MIT Press, Cambridge, MA, 1995.
[35] S. Jarecki, A. Lysyanskaya, Adaptively secure threshold cryptography: introducing concurrency, removing erasures, in: Advances in Cryptology—EUROCRYPT '2000, Lecture Notes in Computer Science, Springer, Berlin, 14–18 May 2000, pp. 221–242.

[36] J. Kilian, E. Petrank, C. Rackoff, Lower bounds for zero knowledge on the internet, in: 39th Annu. Symp. on Foundations of Computer Science, IEEE Press, New York, November 1998, pp. 484–492.
[37] R. Ostrovsky, M. Yung, How to withstand mobile virus attacks, in: Proc. 10th Annu. ACM Symp. on Principles of Distributed Computing, 1991, pp. 51–61.
[38] T.P. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing, in: CRYPTO'91, Advances in Cryptology—CRYPTO '91, Lecture Notes in Computer Science, Vol. 576, 11–15 August 1991, Springer, Berlin, 1992, pp. 129–140.
[39] T. Rabin, A simplified approach to threshold and proactive RSA, in: H. Krawczyk (Ed.), Advances in Cryptology—CRYPTO '98, Lecture Notes in Computer Science, Vol. 1462, Springer, Berlin, 17–21 August 1998, pp. 89–104.
[40] R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signature and public key cryptosystems, Commun. ACM 21 (1978) 120–126.
[41] C.P. Schnorr, Efficient identification and signatures for smart cards, in: CRYPTO'89, Advances in Cryptology—CRYPTO '89, Lecture Notes in Computer Science, Vol. 435, 20–24 August 1989, Springer, Berlin, 1990, pp. 239–252.
[42] A. Shamir, How to share a secret, Commun. ACM 22 (1979) 612–613.
[43] V. Shoup, R. Gennaro, Securing threshold cryptosystems against chosen ciphertext attack, in: Advances in Cryptology—EUROCRYPT 98, Lecture Notes in Computer Science, Vol. 1403, Springer, Berlin, May 1998, pp. 1–16.