Proactive Public Key and Signature Systems

Amir Herzberg* **Markus Jakobsson** † **Stanisław Jarecki** ‡ **Hugo Krawczyk** § **Moti Yung** ¶

Abstract

Emerging applications like electronic commerce and secure communications over open networks have made clear the fundamental role of public key cryptography as a unique enabler for world-wide scale security solutions. On the other hand, these solutions clearly expose the fact that the protection of private keys is a security bottleneck in these sensitive applications. This problem is further worsened in the cases where a single and unchanged private key must be kept secret for very long time (such is the case of certification authority keys, bank and e-cash keys, etc.).

One crucial defense against exposure of private keys is offered by *threshold cryptography* where the private key functions (like signatures or decryption) are distributed among several parties such that a predetermined number of parties must cooperate in order to correctly perform these operations. This protects keys from any single point of failure. An attacker needs to break into a multiplicity of locations before it can compromise the system. However, in the case of long-lived keys the attacker still has a considerable period of time (like a few years) to gradually break the system.

Here we present *proactive public key systems* where the threshold solutions are further enhanced by periodic refreshment of the shared function in such a way that the private key (and its corresponding public key) is kept unchanged for as long as required, yet the breaking of the system requires the attacker to break into several locations in a *short* period of time, e.g during one day or one week. We present such solutions for a variety of discrete log based cryptosystems including DSS and Schnorr signatures, ElGamal-like signatures and encryption, undeniable signatures, and more. We build on previous work on proactive secret sharing and threshold schemes, and develop a general methodology for the combination of many of these systems into secure proactive public key solutions.

*IBM Research, Haifa Scientific Center, *amir@haifa.vnet.ibm.com*
†University of California, San Diego, *markus@cs.ucsd.edu*
‡Massachusetts Institute of Technology, *stasio@theory.lcs.mit.edu*
§IBM T.J. Watson Research Center, *hugo@watson.ibm.com*
¶CertCo, New York, *moti@certco.com*, *moti@cs.columbia.edu*

1 Introduction

The security of traditional public-key systems relies on the secrecy of private keys. These keys are assumed to be inaccessible to any adversary, or in other words, the security of the system relies on the adversary not being able to gain access to the memory locations where these keys are stored. However, in real systems, adversaries – such as corrupted system administrators, accidents and misconfigurations, hackers, trojans and viruses – may eventually gain access to these memories, particularly in the case of software based systems (cf. [ER]). Thus, it is important to protect systems against such strong adversaries, especially when the breaking of such a system represents a significant economic gain for the attacker.

The need to protect against such "memory access" attacks has been recognized in the cryptographic literature long ago, particularly with the introduction of the notion of *secret sharing* (or threshold) schemes [Sh, Bl]. Secret sharing can be used for the distribution of a sensitive private key among different servers. To perform the private operation, e.g., a signature, the key is temporarily reconstructed from a subset of shares. The drawback of this solution is that the reconstruction of the key in one location constitutes a potential point of failure for the entire system, and, in turn, an attractive target for attacks. To solve this problem the notion of *threshold cryptosystems* was suggested, most notably by Desmedt and Frankel (see e.g., [DF]).

In these systems, each server holds a share of the system's private key, and in order for the intended function, e.g., a signature, to be calculated, a large enough subset (a *threshold*) of the servers sharing the secret must collaborate and each produces a partial result. These partial results are then combined through a simple and public (i.e., non-secret) procedure to produce the intended function result. In this case, the individual key shares exist only in the private memory of their owners and are never revealed to other parties. Even repeated operations of the system (e.g., signatures on different messages) will not compromise the shares or require the explicit reconstruction of the shared private key. Thus, in order to compromise the system an attacker needs to penetrate more than a threshold of locations (in particular, this avoids any single point of failure). This methodology has been formalized in [DDFY] through the notion of *function sharing*. To be more useful, threshold cryptosystems need also be robust in the sense of being able to compute the intended function even in the presence of wrong partial results contributed by corrupted parties (cf. [FGY, GJKR2]).

The threshold cryptosystem approach indeed significantly enhances the security of public key cryptosystems. However, in the cases of sensitive and long-lived private keys (e.g., a certification authority key, a bank signature key for e-cash, etc.) the attacker still has a long time to gradually compromise the system. Even a momentary access to a server can expose that server's share for the duration of the life of the key. Although defenses against the spreading of the attack from one share location to others can be mounted (via firewalls, specialized hardware, operating systems, detection tools, etc.) the attacker has a long period of time (e.g., a few years for a certification authority key) to repeatedly try to break into these various locations. Since long-lived public keys will have high economic value, the efforts and money invested by the adversary may be worthwhile.

THE PROACTIVE APPROACH. This approach, intended to defend against repeated attacks by strong and determined adversaries, was introduced by Ostrovsky and Yung [OY], who presented, among other things, information theoretic secret sharing in this model. Herzberg, Jarecki, Krawczyk, and Yung [HJKY] specialized this notion to robust secret sharing schemes which are cryptographic and very efficient. Proactive secret sharing schemes can be used to protect long-lived distributed secrets by periodically refreshing the secret shares in such a way that the exposure of a share to an attacker in a given period of time has no value for that attacker after the refreshment of shares is performed. Most importantly, the shares are renewed but the secret to which they correspond is unchanged. A refreshment phase can be performed after relatively short periods of time, e.g., a day or a week. In that way the adversary that tries to learn the secret needs to corrupt a threshold of locations in a *single and short* period of time. In addition, these refreshment or *update* phases are used to secretly reconstruct any shares that could have been destroyed (accidentally or maliciously), thus ensuring the long term correctness of the shared secret.

The goal of our work here is to apply the proactive notion, and its unique advantages, to threshold public-key cryptosystems in order to prevent their gradual compromise by a determined attacker. In a *proactive threshold cryptosystem* we have periodic *update phases* where key shares are refreshed (and recovered, if found to be corrupted) and *function computation phases* where the parties collaborate to compute the intended function. Even though after each update period the shares are independent of previous shares, the shared function is always the same (e.g, a system that computes signatures will compute them always using the same shared private key, and the same corresponding public key). However, an attacker that intends to break the system is required to learn the key shares of a whole threshold of servers during a *single short period of time*. This is true even if the adversary actively attacks the system both at the update phases and at the function computation phases. Thus, even if the same private key needs to be kept fixed, secret and active for a long time, say several years (e.g., due to the difficulty of globally updating the corresponding public key), the attacker is faced with the task of breaking a threshold of servers in, say, one single week. With the aid of recovery mechanisms applied during the update phases, the correct functioning of the system can be maintained for a long period of time even in the presence of an adversary which breaks at most a threshold of servers in every time period[1]. On the other hand, an attacker that breaks in and learns a share but then loses control of the compromised server cannot take advantage of that knowledge after the share is refreshed. Moreover, this property holds even if the exposure of the shares remains undetected.

An important characteristic of threshold cryptosystems that is preserved in our proactive setting is that of being *transparent* to the user receiving the public-key service. For example, in the case of proactive signatures, a user that requests a signature of a message from the system (e.g., a public key certificate, an electronic coin, etc.), receives the same signature as if it were computed by a single server or requested at another time period. Users need not to be aware of the

[1] We note that the notion of servers recovering after being controlled by an adversary is justified by many mechanisms – virus detection, intrusion detection, connectivity tools, etc.– that are being developed to rid systems of intrusions and attackers and reboot them afresh, cf. [K]

Scheme	DSS [NIST]	AMV [AMV]	Schnorr [Sch]	Undeniable [CvA]	Chaum/Pedersen [CP]
Generation	$s = k^{-1}(m + xr)$	$s = xm - kr$	$e = h(m, r)$ $s = k + xe$	$s = m^x$	$s = m^x$ $\mu = m^k$ $c = h(m, s, r, \mu)$ $d = k + cx$
Signature	(s, r)	(s, r)	(s, r)	s	(s, r, μ, d)
Verification	$r \stackrel{?}{=} g^{ms^{-1}} y^{rs^{-1}}$	$y^m \stackrel{?}{=} r^r g^s$	$e = h(m, r)$ $r \stackrel{?}{=} g^s y^{-e}$	$log_m s \stackrel{?}{=} log_g y$	$c = h(m, s, r, \mu)$ $g^d \stackrel{?}{=} ry^c$ $m^d \stackrel{?}{=} \mu s^c$

Table 1: Signature computation and verification in different signature schemes.

fact that the system is maintained in a distributed and proactive fashion. Namely, in the case of a proactive signature, the signature verification is performed by the standard procedure using only the (single) public key of the scheme.

2 Our Results and their Applications

We first outline the results, their potential applications, and related work.

2.1 Our results

- We define the notion and model of proactive public key systems, with the emphasis on proactive signature systems (sections 3 and 4).
- We describe how the proactive secret sharing mechanism of [HJKY] can be used to transform a broad class of threshold signature schemes based on the hardness of computing discrete logarithms, into proactive signature schemes (section 5):
  - We identify the properties of the proactive secret sharing mechanism of [HJKY] which render the resulting proactive signature scheme provably secure (section 5.1).
  - We specify the conditions which the threshold signature schemes have to meet in order to be "proactivized" with our method (section 5.2).
  - We prove the correctness and security of the general construction yielded by the above design methodology (section 5.3).
- We exemplify our method with proactive solutions to several specific signature functions with known threshold distributed schemes (section 5.4) which we overview below. These examples cover a wide range of applications from general signature schemes to particular methods used in e-cash and other electronic payment solutions. In particular, we show proactive threshold solutions to DSS signatures [NIST, Kra, GJKR1], other ElGamal-type signatures [E], like the one proposed in Agnew, Mullin, and Vanstone [AMV, H], undeniable signatures [CvA, C], Schnorr's signatures [Sch] (as well as its variant for e-cash used in [Br]), and the signature scheme by Chaum and Pedersen used in wallet databases [CP].
- We claim that the above outlined methodology of proactivizing public-key signature schemes can be applied to other public-key systems. For example, we can proactivize the decryption operation in ElGamal cryptosystem [E, DF], and the signature verification and disavowal protocols in undeniable and designated-verifier signature schemes [CvA, C, P1, JY]. This enables proactive version for many systems based on exponentiation with secret keys in known domains. (We stress, however, that this methodology is not "universal", e.g. RSA and factoring based systems are not covered by this work).

EXAMPLES. In table 1 we summarize the signature computation and verification of specific schemes for which we provide proactive solutions. We adopt a standard setting which is common to all these discrete log based problems, in which there are two large primes p, q s.t. $q \mid (p-1)$, and an element g of order q in Z_p^*. The secret key can be any number $x \in Z_q$. Its public verification counterpart is a number $y = g^x \ (mod\ p)$ and the entire public key is a tuple (p, q, g, y). We assume that, for each signature, k is chosen uniformly at random from Z_q, and $r = g^k$ (mod p). All operations on exponents are understood to be modulo q, the others modulo p. An exception is the DSS scheme in which $r = (g^k \bmod p) \bmod q$ and the verification equation is also checked mod$p$ mod q. The message m is in Z_q, and it is typically a hashed value of the original, longer,

message to sign. The function h represents a collision-resistant hash function.

In theorem 5.1, we show that cryptosystems that fall under such a setting and which possess a corresponding secure threshold scheme (the exact conditions are formalized in definition 4.1, section 5.2 and in theorem 5.1 itself), can be proactivized. We discuss the above schemes in detail in section 5.4. Notice that our emphasis is on methodological proactivization of known threshold cryptosystems rather than the design of new threshold schemes.

OUR PROOFS OF SECURITY AND PROTOCOL COMPOSITION. One important aspect of our work is to provide a rigorous proof of security for the general construction that we present. This translates into a proof of security for each of the above specific examples. The core technical part behind these proofs is dealing with the preservation of the security of individual protocols throughout composition. In our case, we need to compose proactive update phases with threshold signatures (or other public key schemes). The natural security requirements from these component protocols when executed in isolation, seem not to be sufficient to prove the security of the composed protocol. We therefore establish the enhanced security properties of the underlying component protocols under which our construction yields secure proactive cryptosystems. In this way we can provide proactive schemes that can be directly built on top of existing threshold solutions for which these composition-driven properties actually hold.

2.2 Applications

Let us list a number of potential applications:

- The importance of long-lived public keys is obvious in the context of a certification authority. Depending on the period of the public key validity, the distribution and frequency of update has to be tuned, so that the assessed risk under the potential attacks is minimized. We stress that in this case, merely changing the key every so often (as usually done to limit the damage caused by key exposures) is not applicable, because the change of public keys in these systems would involve very costly actions of infrastructural nature.
- In electronic commerce schemes, a compromised signature key of a bank could produce huge economic gains to the attacker (i.e., huge losses for the bank). It, therefore, makes sense to invest in protection of a service of this nature.
- Similarly, electronic public notary services, time stamping services, and secure databases may be subject to attacks due to the legal and monetary value of the information that a successful attack can produce.
- Dynamic management via change of key-share holders: This is a crucial application to trust management in public-key systems, which enables dynamic changes in the trust among share-holders. For example, when companies merge or split we can rearrange the way departments' key shares are distributed, as long as at any time period only a relatively small number of changes happens, which is indeed realistic. Using the above proactive approach we can cause certain shareholder to become invalid (by not refreshing their share by the majority) and make new shareholders join (by treating them as existing shareholder who have lost their past shares). This application was first noticed in [FGMY].

2.3 Related work

The adversarial model requiring proactive solutions, where an opponent can repeatedly attack each party, but no more than a threshold of parties at a single time period, was introduced by Ostrovsky and Yung [OY], as the *"mobile adversary"* model. That work deals with the general theory of proactive memory maintenance and computation in the presence of mobile faults in a network of authenticated channels. This first work also left as an open direction of research the existence of practical application of the notion, and this was first answered by Canetti and Herzberg [CH] who designed a practical solution for pseudo-random generators, which can be used for symmetric key cryptosystems (see also [CSH]). Later, [HJKY] introduced the practical notion of *robust proactive secret sharing* and presented a protocol which achieves it (see also [AGY]).

The work we present here has stirred further interest and produced new encouraging recent applications of the notion of proactive public key systems. In fact, two recent manuscripts have presented novel developments building on the work of this paper. First, the notion of proactive cryptosystems as defined here has been applied to RSA [FGMY], thus producing a proactive cryptosystem based on the hardness of factoring. The methodology, techniques, and examples we present here are all based on the difficulty of discrete logarithm problem, whereas the technical challenge with RSA (which has been originally an open question in light of this work) is that the secret key domain must be maintained secure from the share holding servers, throughout. The second manuscript [CHH] employs proactive signature of the type introduced here, and combines it with other tools to produce proactive protocols in the model of

point to point network with unauthenticated channels and no broadcast. (Note that here we assume broadcast and ignore the lower layer of communication needed to support it; this enables us to concentrate on the "cryptographic constructions" level of the protocols.)

3 The Proactive Model

We use the model of [HJKY], where proactive secret sharing was introduced and we adopt it to proactive public key systems. In short, the model is as follows:

SERVERS AND COMMUNICATION MODEL. We employ a group of servers to secret-share a private secret key x. These servers will perform some secret key operation (signature, decryption, etc.) using their sharing of x. The servers are connected to a common broadcast medium C, called communication channel, with the property that messages sent on C instantly reach every party connected to it. We assume that each server has a local source of randomness and that the system is synchronized, i.e., that the servers can access a common global clock. (We note that some instances of the underlying building blocks require highly synchronous communication rounds). How to implement the underlying communication model proactively (based on broadcast and cryptographic tools) is discussed in [HJKY].

TIME. Time is divided into *time periods* which are determined by the common global clock (e.g., a day, a week, etc.). Each time period consists of a short *update phase*, during which the servers engage in an interactive *update protocol*, at the end of which they hold new shares (in fact, new sharing) of the secret x. After the update, there is a *function computation phase*, in which the servers perform the intended secret-key operation using their current sharing of x.

THE MOBILE ADVERSARY. The adversary can corrupt servers at any moment during a time period. If a server is corrupted during an update phase, we consider the server as corrupted during both periods adjacent to that update phase. We assume that the adversary corrupts no more than t out of n servers in each time period, where t must be smaller than $n/2$ (this guarantees the existence of a majority of $t+1$ honest servers at each time).[2] *Corrupting* a server means any combination of learning the secret information of the server, modifying its data, changing its intended behavior, disconnecting it from the communication channel, etc. For the sake of simplicity, we do not differentiate between malicious faults and "normal" server failures (e.g., crashes, power failures etc.).

[2] The exact value of t in a proactive signature sharing protocol is inherited from the underlying threshold signature scheme (see theorem 5.1).

We also assume that the adversary is connected to the broadcast channel C, which means he can hear all the messages and inject his own. He cannot, however, modify messages sent to C by a server that he does not control, nor can he prevent a non-corrupted server from receiving a message sent on C. We assume that the adversary cannot be cut from this communication channel, but we preclude the possibility that the adversary will flood it with messages and thus prevent the servers from communicating. Although this is an attack that can happen in real life, there seem to be no cryptographic ways of preventing it.

Furthermore, we assume the adversary to be *computationally bounded* (and in particular, to be adequately modeled by a polynomial-time Turing machine), so that it cannot break the underlying cryptographic primitives on which we base our design, namely a public-key encryption signature scheme, and a verifiable secret sharing mechanism. We consequently assume that the mobile adversary can attack the system for only polynomially-many time periods.

REMOVAL OF AN ADVERSARY FROM A SERVER. We assume that the adversary intruding the secret-sharing servers is "removable" (e.g., through a reboot procedure) when it is detected. The responsibility for triggering the reboot operation (or other measures to guarantee the normal operation of a server) relies on the system management which gets input from the servers in the network. In addition to regular detection mechanisms (e.g., anti-virus scanners) available to the system management, our protocols provide explicit mechanisms by which a majority of servers always detects misbehaving server and alerts about them. We assume for simplicity that the reboot operation is performed immediately when attacks or deviations from the protocol are detected, and that it takes less time than a duration of a time period.

We remark that the initialization of servers and reboot operations require a minimal level of *trust* in the system management, restricted to installation of correct programs and of public keys used for server-to-server communication. Specifically, no secret information is exposed to the system management.

4 Formalization of Proactive Signature Sharing

Below we formalize the notion of a proactive signature scheme. We stress that an equivalent formalization can be made for other public-key functions, like a decryption function in public-key cryptosystems.

We skip the definitions of unforgeable signature schemes as achieved by the basic signature schemes we

employ, (a strong notion of unforgeability is given in [GMR2]). We also skip the definition of secret sharing [Sh]. We do, however, recollect the definitions of robust threshold signature schemes and of proactive secret sharing, and then we define proactively secure signature schemes.

Robust Threshold Signature Scheme.

A Robust (t,n)-Threshold Signature Scheme is a triple of protocols (Thresh-Key-Gen,Thresh-Sig,Ver), for which there exists an unforgeable non-threshold signature scheme (Key-Gen, Sig, Ver). Thresh-Key-Gen is the distributed key generation protocol performed by the servers $\{P_1,\ldots,P_n\}$. The private output of server P_i contains a value x_i, and the public output of the protocol contains the public key y. Values $(x_1,\ldots,x_n)$ form an implicit (t,n)-threshold secret sharing of the secret key x corresponding to the public key y. The distribution of pairs (x,y) produced by the Thresh-Key-Gen is the same as in the Key-Gen. Thresh-Sig is the distributed version of the signature protocol Sig, performed by servers $\{P_1,\ldots,P_n\}$ on their private inputs $\{x_1,\ldots,x_n\}$, and the public input a message m and the public key y.

Definition 4.1 *Let* (Thresh-Key-Gen,Thresh-Sig,Ver) *be a triple of protocols; we call it a* **Robust** (t,n)**- Threshold Signature Scheme** *if the following conditions hold:*

- Unforgeability, *i.e., no malicious adversary who corrupts at most t servers can produce the signature on any new (i.e., previously unsigned) message m, given the view of the protocol* Thresh-Key-Gen *and of the protocol* Thresh-Sig *on input messages which the adversary adaptively chose.*
- Robustness, *i.e., even in the presence of an adversary who corrupts t servers, both* Thresh-Key-Gen *and* Thresh-Sig *complete successfully, i.e. after performing* Thresh-Key-Gen *once, every time some message m is submitted for signature as a public input,* Thresh-Sig *results in a public output sig, s.t.* Ver$(sig,m,y)=$ *"correct".*

Proactive Secret Sharing.

We adapt the definition of proactive secret sharing from [HJKY] to the case of distributed generation and sharing of a random value without a trusted server, as opposed to sharing a number provided by some trusted center. Such a definition is useful for us, because we use secret sharing in threshold signature sharing, which (as defined above) is initialized without a trusted center. A proactive (t,n)-threshold secret sharing is a triple of protocols (Init,Update,Reconstruct) performed by a group of n servers. Init is the distributed secret-sharing initialization protocol, in which the private output of server P_i contains x_i, s.t. values $\{x_1,\ldots,x_n\}$ form a secret-sharing of some secret x. After the initialization, at the beginning of every *time period* (see the model of time in section 3), the servers perform the Update protocol, in which the private input of P_i is x_i, and the private output of P_i is x'_i, s.t. values $\{x'_1,\ldots,x'_n\}$ form a new secret-sharing of the same secret x. Finally, Reconstruct is a secret reconstruction algorithm, where the private input of P_i is its current share x_i, and the public output is the secret x.

Definition 4.2
Let (Init,Update,Reconstruct) *be a triple of protocols; we call it a* **Proactive** (t,n)**- Threshold Secret Sharing Scheme** *if the following conditions hold:*

- Secrecy, *i.e., the* mobile *adversary who corrupts at most t servers* in each time period, *given the view of the protocol* Init *and the view of multiple consecutive protocols* Update, *cannot learn any more about the secret x, then what he can learn from the public information. (This notion of secrecy is usually referred to as* semantic security.*)*
- Robustness, *i.e., even in the presence of a mobile adversary who corrupts t servers in each time period, after a single* Init *and multiple* Update *protocols, if $t+1$ servers initialize the* Reconstruct *protocol, its public output is the original secret x which was shared in the* Init *protocol.*

Proactive Signature Sharing.

We define a restricted notion of proactive (t,n)-threshold signature schemes, which are constructed from some existing robust (t_1,n)-threshold signature scheme (Thresh-Key-Gen,Thresh-Sig,Ver) and some proactive (t_2,n)-threshold secret sharing scheme (Init,Update,Reconstruct), where $t=\min(t_1,t_2)$. This is not the most general definition, since in principle proactive function sharing could be achieved in some other way. Yet, the schemes which we present in this paper all have this form, so this definition is the most useful here. Such a proactive secret sharing scheme contains four protocols (Pro-Key-Gen,Update,Thresh-Sig,Ver), where Pro-Key-Gen combines the features of Init and Thresh-Key-Gen, Update is performed at the beginning of every time period as in the proactive secret sharing scheme, and Thresh-Sig and Ver are performed in the same way as in the threshold signature scheme.

Definition 4.3 *We call a 4-tuple of protocols* (Pro-Key-Gen,Update,Thresh-Sig,Ver) *a* **Proactive** (t,n)**-Threshold Signature Scheme** *if the following conditions hold:*

- Unforgeability, *i.e., no* mobile *adversary which corrupts at most t servers* in each time period *can*

produce the signature on any new (i.e., previously unsigned) message m, given the view of the protocol Pro-Key-Gen *and of multiple consecutive protocols* Update, *and of many protocols* Thresh-Sig, *which were performed during the time periods and on input messages which the adversary adaptively chose.*

- Robustness, *i.e., even in the presence of a mobile adversary who corrupts t servers in each time period, after a single* Pro-Key-Gen *protocol and multiple* Update *protocols, every time*[3] *some message m is submitted to the servers for signature,* Thresh-Sig *results in a public output sig, s.t.* $\mathsf{Ver}(sig, m, y) =$ *"correct".*

The above definitions deal with *robust* threshold signature schemes. Similar definitions could be formed for non-robust (i.e., "regular") threshold signatures, which, combined with proactive secret sharing, would then result in non-robust proactive signature schemes. The difference would be in the power of the adversary these schemes would be secure against: The regular (t, n)-threshold signature schemes are secure only against an adversary who *spies* on up to t servers, but does not *corrupt* them (as in the unforgeability requirement of definition 4.1). Similarly, the non-robust proactive signature scheme would be secure against a *mobile but not fully-malicious* adversary who can only spy on up to t servers in every time period. This notion is perhaps less interesting and we ignore it in the rest of the paper.

5 The Methodology for Proactivization of Threshold Signature Schemes

We present a methodology of proactivizing robust (t, n)-threshold signature schemes of certain type (for any $t \leq \lfloor \frac{n}{2} \rfloor$). Again, we discuss robust "signatures" for concreteness and since most of our examples are signature schemes. We also present application of the methodology to decryption function in public-key cryptosystem, and to the function of verification/repudiation of signature validity in the undeniable signature scheme.

We first outline the important properties of the (Init,Update,Reconstruct) proactive secret sharing scheme we will use. Next, we list the requirements on a threshold signature schemes (Thresh-Key-Gen,Thresh-Sig,Ver) which enable us to proactivize them with the above proactive secret sharing. Then, we show how to combine Thresh-Key-Gen and Init into Pro-Key-Gen and we prove that the resulting scheme (Pro-Key-Gen,Update,Thresh-Sig,Ver) is a proactive signature scheme. Lastly, we present examples of signature schemes that are proactivizable with this method, and we remark on public-key functions other then signatures which can be proactivized in the same way.

[3] We assume that the messages submitted for signatures during update phases will simply wait till the Update protocol is done, and then will trigger the Thresh-Sig protocol.

5.1 Properties of the Proactive Secret Sharing Protocol (PSS)

Below we outline the (Init,Update,Reconstruct) ($t = \lfloor \frac{n}{2} \rfloor, n$)-threshold proactive sharing protocol (let us call it PSS) presented in [HJKY]. Note that this scheme can be adapted to any t smaller than $\lfloor \frac{n}{2} \rfloor$. We also state the properties of that protocol which allow us to use it as a building block in proactive *function sharing* schemes. Note that we modify the original Init protocol slightly, so that the secret-shared value x is picked at random by the servers, and not handed over to them by some third party. As described in section 3, the lifetime of a proactive secret sharing scheme is slotted into *time periods*, which are separated by relatively short *update phases* during which protocol Update re-randomizes the the secret sharing and recovers any corrupted information.

- Init: Protocol Init is the distributed verifiable secret sharing protocol of Feldman[F] (see also [P2] for a simple presentation of that protocol), where each active party serves as a distributor and each share x_i (of party i) results from adding up all the sub-shares given to party i by each of the distributors (including i itself). Before being added in, the validity of these sub-shares is verified.

  Having a security parameter as an input, protocol Init outputs the description of the space over which the secret sharing will be performed, namely it creates a pair p, q of big primes s.t. $q \mid (p-1)$, and an element g of order q in Z_p^*. It also gives each server P_i a share $x_i \in Z_q$ as its private output, and $\mathrm{Pub} = \{g^{x_1}, \ldots, g^{x_n}, g^x\}$ as a public output.[4]

  Let $\mathcal{S}$ be the space $\mathcal{S} = (Z_q)^n \times (Z_p^*)^{(n+1)}$. For every $x \in Z_q$, we will denote by $\mathcal{S}_x$ a probability distribution in $\mathcal{S}$ of elements $I = (\{x_1, \ldots, x_n\}, \{y_1, \ldots, y_n, y\})$ s.t. $\{x_1, \ldots, x_n\}$ form a random $(\lfloor \frac{n}{2} \rfloor, n)$-threshold polynomial secret sharing (see [Sh]) of a number x s.t. $y = g^x$, and $y_i = g^{x_i}$ for every $i \in \{1 \ldots n\}$.

  In the above notation we can summarize protocol Init as picking x uniformly in Z_q, and then picking a random $I_0 = (\mathrm{Pri}_0, \mathrm{Pub}_0) = (\{x_1, \ldots, x_n\}, \{y_1, \ldots, y_n, y\})$ from the probability distribution $\mathcal{S}_x$ and outputting it as a private and public output respectively.

[4] All exponentiations are performed in Z_p^*.

- Reconstruct: Protocol Reconstruct is the same verifiable (i.e., robust) secret reconstruction protocol as in Feldman's VSS [F]. For every $I \in \mathcal{S}_x$, Reconstruct$(I) = x$, even if up to t servers are compromised by the adversary.[5]

- Update: Protocol Update is performed by all the (active) servers, each acting as a distributer and as receiver, at the beginning of each time period. On input $I_{i-1} = (\mathrm{Pri}_{i-1}, \mathrm{Pub}_{i-1})$, even in the presence of an adversary who compromises up to t servers in both the $(i-1)$st and the ith time periods, protocol Update outputs I_i, which is picked anew at random from the probability distribution $\mathcal{S}_x$. This property of Update leads to the robustness of the whole proactive protocol PSS, because then for every time period i, Reconstruct$(\mathrm{Pri}_i, \mathrm{Pub}_i)$ outputs x even in the presence of a mobile adversary who compromises up to t servers in every time period.

The Init and Update protocols of PSS above achieve secrecy and robustness in the strong sense of *semantic security relative to any auxiliary information*, which is formalized below. This property makes the PSS protocol secure and robust, and we will use it to prove robustness and unforgeability of the proactive signature schemes based on PSS (theorem 5.1).

Definition 5.1 *We call protocols* Init *and* Update semantically secure relative to any auxiliary information *if for every prior knowledge κ, no computationally-bounded machine on the input κ and the complete view of a t-threshold mobile adversary who corrupts up to t servers every time period, can compute anything more (with non-negligible probability) than a polynomial-time machine which takes as inputs κ, the public information p, q, g, g^x, and the secret shares of up to t servers in each time period.*

We also notice that because of the use of Feldman's Verifiable Secret Sharing mechanism [F], the robustness of the PSS protocol is not subject to computational bounds on the adversary. The t-threshold mobile adversary could not disrupt the PSS protocol even if he knew all the private information of every server, and in particular the secret x itself.

For more detailed formalization and proofs of the above properties of this protocol, we refer the reader to [J].

[5] Of course, the inputs and outputs to the servers controlled by the adversary are arbitrary. The specified distribution describes the inputs and outputs only of the honest parties.

5.2 Our Requirements for Proactivization of Threshold Signature Schemes

We identify the following properties of a robust (t, n) threshold signature scheme (Thresh-Key-Gen,Thresh-Sig,Ver) as sufficient conditions (when satisfied simultaneously) for proactivization with the help of the PSS protocol described above.

1. Discrete-Log Based Signature Scheme. The original signature scheme is based on the hardness of computing the discrete log. Specifically, the message and key spaces are picked like in the Init protocol of PSS: p, q are big primes s.t. $q \mid (p-1)$, g is an element of order q in Z_p^*, the private signature key x is picked uniformly from Z_q and the public key is $y \in Z_p^*$ s.t. $g^x = y$.
2. Shamir Secret-Sharing of the Key. The private outputs $\{x_1, \ldots, x_n\}$ of the Thresh-Key-Gen protocol form a random (t, n)-threshold Shamir' polynomial secret sharing of the secret signature key x.
3. Verification Information. In addition to the private shares $\{x_1, \ldots, x_n\}$ and the public data $\{p, q, g, y\}$, the Thresh-Key-Gen protocol can optionally distribute $\{g^{x_1}, \ldots, g^{x_n}\}$ as a public information. This requirement is necessary to claim robustness of the resulting proactive signature scheme.
4. Simulatability of the Threshold Signature Protocol. We require that the adversary's view of that protocol be *simulatable*, in the sense of definition 5.2 below.

The first two requirements are obvious. If the threshold signature scheme is to perform correctly in every function computation phase, every run of proactive Update protocol must output private and public information in such a way, so that it would look just as if it was picked by the Thresh-Key-Gen protocol of the threshold signature scheme. Requirements 1 and 2 specify the threshold signature scheme (Thresh-Key-Gen,Thresh-Sig,Ver) to achieve this necessary equivalence of protocol Thresh-Key-Gen and protocol Init from PSS.

The third requirement plays the same role. To claim robustness of the Thresh-Sig protocol in every phase, we have to make sure that the Update protocol of PSS produces everything that Thresh-Sig needs (which in turn must be the same as what Thresh-Key-Gen produces). But we know that Update protocol outputs exactly $I = (\mathrm{Pri}, \mathrm{Pub}) = (\{x_1, \ldots, x_n\}, \{g^{x_1}, \ldots, g^{x_n}, g^x\})$ in probability distribution $\mathcal{S}_x$. Hence we have to constrain the

additional public information output by Thresh-Key-Gen to $\{g^{x_1}, \ldots, g^{x_n}\}$. Intuitively, the threshold signature scheme cannot require any more private or public data (produced by Thresh-Key-Gen and used by Thresh-Sig) than can be proactively maintained by PSS.

The simulatability property (requirement four) [GMR1] will allow us to claim that the adversary does not gain any knowledge by repeated executions of Thresh-Sig even when the underlying secret sharing of the signature key x is being re-randomized by the proactive secret sharing mechanisms:

Definition 5.2 *We call a threshold signature protocol* Thresh-Sig simulatable *if there exists a polynomial-time machine, which from the public output of* Thresh-Key-Gen*, from the resulting signature sig on m, and from up to t values x_i learned by the adversary from the corrupted servers, can generate a probability distribution that is indistinguishable from the view of the adversary during the execution of the* Thresh-Sig *that generates sig.*

Remark: Depending on the setting, we may require the distribution of the simulated view to either be identical to or computationally indistinguishable from (given some realistic hardness assumption) the distribution of the real view.

5.3 The resulting Proactive Signature Scheme

Proactive Signature Key Generation Protocol. The Pro-Key-Gen protocol in our proactive signature sharing is the same protocol as the Init of the PSS (section 5.1). From requirements 1,2 and 3 of Thresh-Key-Gen of the threshold signature scheme, it follows that the Init protocol of PSS will perform all the functions required by the Thresh-Key-Gen. The only difference between Init and Pro-Key-Gen, is in the terminology: The secret number x randomly selected in Z_q will be called a *secret signature key*, and the value $y = g^x$ which is a part of the public output will be called a *public verification key* of the proactive signature sharing scheme.

Theorem 5.1 *If the following conditions hold:*

- (Init,Update,Reconstruct) *is a proactive secret sharing protocol PSS (as described in section 5.1)*
- (Thresh-Key-Gen,Thresh-Sig,Ver) *is a robust (t, n)-threshold signature scheme and is simulatable (satisfies all the requirements specified in section 5.2)*
- Pro-Key-Gen *is a proactive key generation, same as* Init *(as described in the preceding paragraph).*

then the resulting scheme (Pro-Key-Gen,Update,Thresh-Sig,Ver) *is a proactive (t, n)-threshold signature scheme.*

Proof: (Sketch). Following definition 4.3, we need to prove that the resulting proactive signature scheme is unforgeable and robust. In our proofs we show that if there exist an attack against our proactive signature scheme, then there will also exist an attack against either the PSS proactive secret sharing scheme, or the robust threshold signature scheme (contradicting their respective security properties).

Robustness.

As discussed in section 5.1, the robustness property of PSS is achieved by the fact that protocol Update, on input $I_{i-1} \in \mathcal{S}_x$, outputs I_i which is picked at random from the same probability distribution $\mathcal{S}_x$. As noted there, this robustness property is unconditional, namely, independent of the computational power of the adversary. Therefore, the robustness property of the Update protocol is preserved through its repeated executions in protocol (Pro-Key-Gen,Update,Thresh-Sig,Ver), because, due to the unconditional nature of the robustness property, any extra knowledge obtained by an attacker during the executions of the proactive signature scheme is of no value in an attempt to disrupt the execution of the Update protocol.

Knowing that $I_k \in \mathcal{S}_x$, we want to show that the t-threshold mobile adversary cannot disrupt the Thresh-Sig protocol in any time period. Let us assume then, that the adversary manages to disrupt the Thresh-Sig protocol in the k-th time period. Since $I_k \in \mathcal{S}_x$, this means that there exists an algorithm A, which, having the view $View$ of the t-threshold mobile adversary of the (Pro-Key-Gen,Update,Thresh-Sig,Ver) protocol for $k-1$ periods, and an access to the signing oracle O_{sig} (i.e., $\forall m \in Z_q, O_{sig}(m) = s$, s.t. $\mathsf{Ver}(y, m, s) =$"correct"), can disrupt the Thresh-Sig protocol on a random $I \in \mathcal{S}_x$. We will show that this implies the existence of an algorithm A^*, which disrupts the Thresh-Sig protocol with $View^*$ of the t-threshold adversary attacking the system *only* during the k-th time period. Since $I_k \in \mathcal{S}_x$, by requirements 1,2 and 3 on (Thresh-Key-Gen,Thresh-Sig,Ver), the success of A^* would contradict the assumption that (Thresh-Key-Gen,Thresh-Sig,Ver) is a robust threshold signature scheme.

To prove this reduction, we need to argue that $View$ can be simulated by A^* from $View^*$, and from the access to the oracle O_{sig}. $View$ contains, first of all, the sets of up to t shares x_i of x from each of the $k-1$ time periods. But these shares are statistically independent from x, by the properties of Shamir's secret sharing. Next, from the public information $p, q, g, y \in View^*$ and the k sets of t shares from every time period, A^*

can simulate all the information in $View$ pertaining to the Update protocols, by virtue of semantic security of Update (see definition 5.1). Having the same input plus the access to the oracle O_{sig}, A^* can also simulate the part of $View$, which pertains to the Thresh-Sig protocols in rounds 1 to $t-1$ that A participated in, by the virtue of requirement 4 on the threshold signature scheme (Thresh-Key-Gen,Thresh-Sig,Ver). This shows that A has no advantage over A^* and concludes the proof of robustness of (Pro-Key-Gen,Update,Thresh-Sig,Ver).

UNFORGEABILITY.

As shown above, at the beginning of each time period the private and public inputs I of the servers have the same probability distribution as if created by Thresh-Key-Gen(the only possible difference is that Thresh-Key-Gen creates $\{g^{x_1}, \ldots, g^{x_n}\}$ only optionally, but they can anyway be computed from the t "secret shares" and the public value g^x that we let that adversary see). Let us assume that the t-threshold mobile adversary can forge the signature in time period k, i.e., that there is a machine A, which on input $View$ of the whole (Pro-Key-Gen,Update,Thresh-Sig,Ver) protocol (with Thresh-Sig protocol running on the messages of the adversary's choice) through time periods 1 to $k-1$, can produce a signature on a new message. We will show that this implies that there is a machine A^* which having only the view of the Thresh-Sig protocol running in the k-th time period (but again on the messages of the adversary's choice), can also produce the signature on the same new message.

This reduction follows from what we proved in the robustness part above, i.e., from the fact that A^* can produce the input to A, and hence could compute whatever A computes. ■

Remark: Note that in our proactivization methodology we only maintain the original notion of security of the underlying signature scheme. The formulation in definitions 4.1 and 4.3 of unforgeability of threshold and proactive signature scheme as resistant to the *adaptive chosen message attack*, was not essential in the above proof (and was used as an example). Similar definitions and the same proof could be made for other notions of security of signature schemes (for examples we refer the reader to [GMR2]).

5.4 Examples of Proactivizable Threshold Signature Schemes

The following corollary lists the signature schemes which can be transformed into proactive cryptosystems using the methodology presented above.

Corollary 5.2 *The following signature schemes are proactivizable:*

1) DSS [NIST, Kra, GJKR1]
2) AMV-Harn [AMV, H]
3) Schnorr [Sch] (and its blind variant [Br])
4) undeniable signatures [CvA, C]
5) Chaum/Pedersen [CP]

The proof of this Corollary will rely on exhibiting for each of the above signature schemes, a threshold version which meets the robustness and simulatability requirements as specified in section 5.2. Some of these versions exist in the literature. In particular, [GJKR1] present a detailed scheme for threshold DSS [NIST, Kra], and show the unforgeability, robustness and simulatability conditions required for our results to hold. Similar mechanisms and arguments can be applied to show the security of the threshold version of Schnorr's signature [Sch], AMV-signatures [AMV, H], undeniable signatures [CvA, C], and the signatures by Chaum/Pedersen [CP]. We refer to [GJKR1] for the methodology of proving the simulatability requirement for these schemes.

Proactivization of Other Public-Key Systems. As mentioned before, we used signature schemes only as an example of discrete-log based public key systems. The proactive secret sharing protocol PSS could be used to proactivize the threshold sharing of El Gamal encryption function [E] proposed by Desmedt and Frankel [DF], where message m is encrypted using the recipient's public key $y = g^x$, by selecting a random number $\rho \in Z_q$ and calculating $(b, c) = (g^\rho, y^\rho m)$. The receiver decrypts by calculating $m = c/b^x$. In fact, their scheme can be made robust [FGY].

We note that the properties that our concrete methodology requires from El Gamal is the security of the underlying encryption, i.e., El Gamal ciphertext security given known/chosen plaintexts (and their ciphertexts). This is different than the unforgeability requirement in the case of a signature scheme. Naturally, we require simulatability of the threshold encryption scheme given such ciphertext, cleartext pairs.

Similarly, PSS can be employed to proactivize the threshold signature verification protocols [P1, JY] for undeniable signatures [CvA, C].

References

[AMV] G. Agnew, R.C. Mullin, S. Vanstone, *Improved digital signature scheme based on discrete exponentiation.* Electronics Letters, v. 26, 1990, pp. 1024-1025.

[AGY] N. Alon, Z. Galil and M. Yung, *Dynamic-Resharing Verifiable Secret Sharing against Mobile Adversary*. 3-d European Symp. on Algorithms (ESA)'95. Lecture Notes in Computer Science Vol. 979, P. Spirakis ed., Springer-Verlag, 1995, pp. 523-537.

[Bl] R. Blakley, *Safeguarding Cryptographic Keys*, FIPS Con. Proc (v. 48), 1979, pp. 313–317.

[BCDP] J. Boyar, D. Chaum, I. Damgård and T. Pedersen, *Convertible Undeniable Signatures*, *Advances in Cryptology – Crypto 90 Proceedings*, Lecture Notes in Computer Science Vol. 537, A. J. Menezes and S. Vanstone ed., Springer-Verlag, 1990, pp. 189-205.

[Br] S. Brands, *Untraceable Off-line Cash in Wallet with Observers*, *Advances in Cryptology – Crypto 93 Proceedings*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed., Springer-Verlag, 1993, pp. 302-318.

[CH] R. Canetti and A. Herzberg, *Maintaining Security in the Presence of Transient Faults*, *Advances in Cryptology – Crypto 94 Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994, pp. 425-438.

[CHH] R. Canetti, S. Halevi and A. Herzberg, *Maintaining Authentication in Secure Communication*, in preparation.

[CSH] C. S. Chow and A. Herzberg, *Network Randomization Protocol: A Proactive Pseudo-Random Generator*, the Fifth Usenix Security Symposium, June 1995, pp. 55-64

[C] D. Chaum, *Zero-knowledge undeniable signatures*, *Advances in Cryptology – Eurocrypt 90 Proceedings*, Lecture Notes in Computer Science Vol. 473, I. Damgård ed., Springer-Verlag, 1990, pp. 458-464

[CvA] D. Chaum, H. van Antwerpen, *Undeniable Signatures*, *Advances in Cryptology – Crypto 89 Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989, pp. 212-216

[CP] D. Chaum, T. P. Pedersen, *Wallet Databases with Observers*, *Advances in Cryptology – Crypto 92 Proceedings*, Lecture Notes in Computer Science Vol. 740, E. Brickell ed., Springer-Verlag, 1992, pp. 89-105

[CGMA] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch, *Verifiable Secret Sharing and Achieving Simultaneous Broadcast*, Proceedings *of the* 26th *Symposium on Foundations of Computer Science*, IEEE, 1985, pp. 335-344.

[DDFY] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, *How to Share a Function Securely*, ACM Proceedings *of the* 26th *Annual Symposium on Theory of Computing*, ACM, 1994, pp. 522-533.

[DF] Y. Desmedt and Y. Frankel, *Threshold cryptosystems*, *Advances in Cryptology – Crypto 89 Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989, pp. 307-315.

[ER] M. Eichin and J. Rochlis, *With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988*, IEEE Sym. on Security and Privacy, 1989, pp. 326-343.

[E] T. El Gamal, *A Public key cryptosystem and a signature scheme based on discrete logarithm*, IEEE Trans. on Information Theory 31, 465-472, 1985.

[F] P. Feldman, *A Practical Scheme for Non-Interactive Verifiable Secret Sharing*, Proceedings *of the* 28th *Symposium on Foundations of Computer Science*, IEEE, 1987, pp.427-437

[FGY] Y. Frankel, P. Gemmel and M. Yung, *Witness Based Cryptographic Program Checking and Robust Function Sharing*. Proceedings *of the* 28th *Annual Symposium on Theory of Computing*, ACM, 1996, pp. 499-508.

[FGMY] Y. Frankel, P. Gemmel, P. MacKenzie and M. Yung. *Proactive RSA*, manuscript.

[GJKR1] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, *Robust Threshold DSS Signatures*, *Advances in Cryptology – Eurocrypt 96 Proceedings*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996, pp. 354-371.

[GJKR2] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, *Robust Threshold RSA*, *Advances in Cryptology – Crypto 96 Proceedings*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996, pp. 157-172.

[GMR1] S. Goldwasser, S. Micali and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems*, Siam J. on Computing, 18(1) (1989), pp. 186-208.

[GMR2] S. Goldwasser, S. Micali and R. Rivest, *A Digital Signature Scheme Secure against the Chosen-Message Attack*, Siam Journal on Computing, Vol. 17, 2 (1988), pp. 281-308.

[H] L. Harn, *Group oriented (t,n) digital signature scheme*, IEEE Proc.-Comput.Digit.Tech. Vol. 141 No.5 September 1994, pp. 307-313.

[JY] M. Jakobsson, M. Yung, *Proving Without Knowing: On Oblivious, Agnostic and Blindfolded Provers*, *Advances in Cryptology – Crypto 96 Proceedings*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996, pp. 186-200.

[J] S. Jarecki, *Proactive Secret Sharing and Public Key Cryptosystems*, Master thesis, MIT, 1996.

[HJKY] A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung, *Proactive Secret Sharing, or: how to cope with perpetual leakage*, *Advances in Cryptology – Crypto 95 Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995, pp. 339-352.

[K] P.A. Karger, *Limiting the Damage Potential of Discretionary Trojan Horses*, IEEE Sym. on Security and Privacy, 1987, pp. 32-37.

[Kra] D. W. Kravitz, *US patent* 5,231,668, July 1993.

[NIST] National Institute for Standards and Technology, *Digital Signature Standard (DSS)*, Federal Register, vol 56, no 169, 20 Aug. 1991.

[OY] R. Ostrovsky and M. Yung, *How to withstand mobile virus attacks*, Proc. of the 10th ACM Symposium on the Principles of Distributed Computing, 1991, pp. 51-61.

[P1] T.P. Pedersen, *Distributed Provers with Applications to Undeniable Signatures*, *Advances in Cryptology – Eurocrypt 91 Proceedings*, Lecture Notes in Computer Science Vol. 547, D. Davies ed., Springer-Verlag, 1991, pp. 221-242.

[P2] T.P. Pedersen, *A threshold cryptosystem without a trusted party*, *Advances in Cryptology – Eurocrypt 91 Proceedings*, Lecture Notes in Computer Science Vol. 547, D. Davies ed., Springer-Verlag, 1991, pp. 129-140.

[P3] T.P. Pedersen, *Non-interactive and information theoretic secure verifiable secret sharing*, *Advances in Cryptology – Crypto 91 Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991, pp. 129-140.

[RSA] R. Rivest, A. Shamir and L. Adleman, *A Method for Obtaining Digital Signature and Public Key Cryptosystems*, Comm. of ACM, 21 (1978), pp. 120-126.

[Sch] C. P. Schnorr, *Efficient Signature Generation for Smart Cards*, *Advances in Cryptology – Crypto 89 Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989, pp. 239-252

[Sh] A. Shamir, *How to share a secret*, Comm. of ACM, 22 (1979), pp. 612-613.