Proactive DSA Application and Implementation

Timothy Draelos, Victoria Hamilton, and Gabi Istrail
Sandia National Laboratories
Albuquerque, NM 87185
tjdrael@sandia.gov vahamil@sandia.gov ggistra@sandia.gov

Abstract

Data authentication as provided by digital signatures is a well-known technique for verifying data sent via untrusted network links. Recent work has extended digital signatures to allow jointly generated signatures using threshold techniques. In addition, new proactive mechanisms have been developed to protect the joint private key over long periods of time and to allow each of the parties involved to verify the actions of the other parties. In this paper, we describe an application in which proactive digital signature techniques are a particularly valuable tool. We describe the proactive DSA protocol and discuss the underlying software tools that we found valuable in developing an implementation. Finally, we briefly describe the protocol and note difficulties we experienced and continue to experience in implementing this complex cryptographic protocol.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

1. Introduction

The increasing use of digital information in a multitude of applications continues to drive the need for effective data surety measures to offer security in the form of data integrity, authenticity, and privacy. Powerful cryptographic techniques have become public in recent years and are finding their way into more and more commercial uses. This paper describes the marriage of a distributed cryptographic signature mechanism and an application that supports distributed and mutually distrusting parties.

In public-key systems, a private signature key usually is held by a single user. This requires all other users to trust this user and provides a single point of attack for an adversary. In addition, an adversary has the lifetime of the key to gain access to its value. With the advent of secret-sharing or threshold mechanisms [S79] [DF89], it became possible to distribute shares of the key among many users. Now trust can be distributed. No single user needs hold the key. An adversary would have to corrupt a threshold of users to gain access to their shares of the key in order to reconstruct the key. However, the adversary still has the lifetime of the key to obtain and/or corrupt these shares. For systems that require long-lived keys, threshold mechanisms are not sufficient to provide adequate security.

The development of proactive mechanisms addresses this problem [HJKY95] [HJJKY97]. Proactivized threshold techniques provide a means for the key shares to be updated on a regular basis. Any information that an adversary learns or destroys in one time period is of no consequence in the next time period even though the global secret itself does not change. Only the shares or distribution of the secret (or key) change. To corrupt the system, an adversary must be able to corrupt a threshold number of shareholders within a very limited time period. This time period should be much shorter than the lifetime of the key. Proactivized threshold systems and threshold systems in general have the added advantage that they are transparent to the signature verifier. A signature generated by a proactive threshold mechanism and a signature distributed by a single user are indistinguishable as long as the private key is the same.

However, proactivized threshold signature schemes are quite computationally intensive and for even small systems, require extensive communications. They rely on many cryptographic building blocks which are not required for standard public-key signature algorithms. Because of the computational and

communication costs involved, appropriate applications should be chosen with care. Nevertheless, there are some interesting advantages to undertaking such development given that an appropriate application exists. Since we must support a variety of cryptographic mechanisms for a variety of applications on a variety of platforms with very limited staff, we maintain a portable cryptographic toolkit with algorithm/application specific enhancements. This toolkit has allowed us to implement the work described in this paper more quickly, while permitting the development of additional tools for the toolkit. It is our hope that these tools will be of value to unrelated work in the future.

Besides being computationally intensive, the development effort was quite complex for a variety of reasons. Most of the references used during development were fairly theoretical. Often, developing a practical implementation was not straightforward. One or two sentence comments in a reference could and did expand greatly upon implementation. This is particularly true in the area of the communication architecture required for secure implementation. In addition, the cryptographic mechanism implemented was actually composed of several independent cryptographic mechanisms. These components had to be integrated in such a way so that system security was enhanced, not compromised.

The following sections describe the project and work required to develop our solution. Section 2 describes the application area which makes use of this work. Section 3 specifies the implementation and current status of the development effort.

2. Application

New and existing weapons and nuclear materials treaties require that signatories be able to monitor each party's activities. This monitoring is often performed remotely by unattended automated sensing systems within the borders of a country which may or not wish to make a substantial investment in resources to evade detection of non-compliance. A typical monitoring system involves collection of data by a sensing system and communication to one or more locations where it is analyzed to detect treaty violations. Sometimes the system itself is intelligent enough to detect obvious events and register an alarm. In addition to sensor data communicated out of the sensor system, commands can be sent to the sensor system to control various operational conditions (see Figure 1).

Figure 1. Basic monitoring system.

This simplistic description of the application should immediately raise questions regarding the security of such a system. Sensor data received by remote review systems must have integrity if compliance is to be credibly assessed. In multilateral treaties (the typical case), integrity must be proven to multiple parties making a public key solution appealing. A solution to the sensor data integrity problem is to apply digital signatures to all sensor data. Here, single key/single signature mechanisms are perfectly appropriate. However, for commands sent to the sensor system, multiparty authorization can be valuable, thereby motivating the use of threshold protocols. Of course, with any kind of public key solution, certification of the public keys must be performed by some certification authority (CA) (see Figure 2).

DISCLAIMER

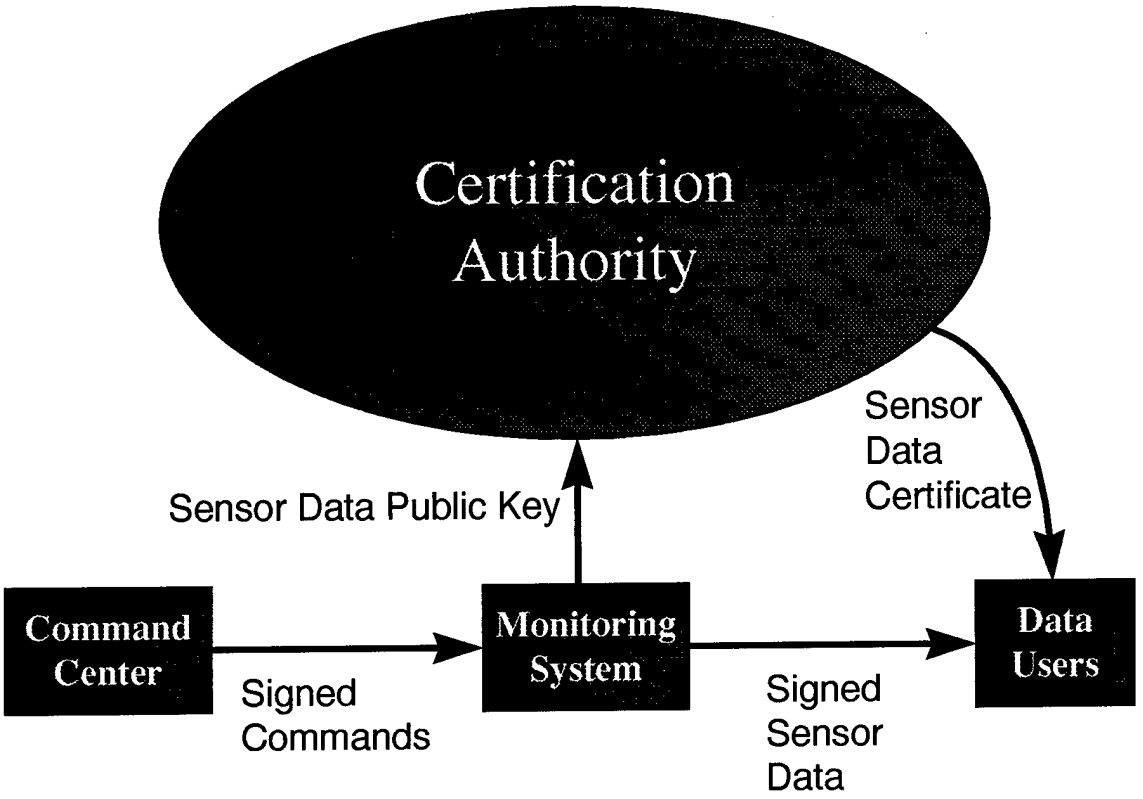

Figure 2. Authenticated monitoring system.

The most important feature of the international treaty monitoring environment that warrants the use of proactive technologies is the trust model. In this environment, no single entity will be trusted to a satisfactory degree by all parties. However, combinations or groups of parties will likely offer the trust necessary to perform cryptographic functions (e.g. CA operations, command authentication). Moreover, the mutual distrust among parties can in fact be an advantage with the advent of trust-distribution solutions. For example, with a group of highly distrusting parties involved in a threshold scheme, a user or beneficiary of the group need not trust any of the parties, but rather trust that a sufficient number of them will not collude to compromise the system. In fact, this is commonly the case in the international treaty monitoring environment.

The following features must be available to a community of mutually distrusting parties in need of trusted data products.

1. No trusted third party required (i.e., no trusted dealer) and no party can be trusted more than any other.
2. Participants in the threshold scheme can be physically separated.
3. All data transactions are verifiable.
4. The group activity can withstand dishonest shareholders.
5. The group key can be protected for a long period of time.

In some instances, it can be advantageous to require multiple parties for proper authorization of commands. The sensor system would simply need to store the group's public key for command verification. Here, the problem can get somewhat complicated when political issues must be considered. For example, the country responsible for the sensor system, commonly called a host country, generally demands participation in the authorization of all commands and will not tolerate denial of commands by a small subset of other parties. This kind of situation calls for grouping of participating parties in potentially complicated ways, possibly not a simple threshold.

The implementation of any cryptographic technology in the international treaty monitoring environment must always be

1. unclassified,
2. exportable,
3. and inspectable.

In other words, the implementation must be completely open, implying that source code must be provided in the case of software products.

3. Implementation

In this section we describe the algorithm and implementation used to support the international treaty monitoring application arena. Some of the most difficult implementation issues are not associated with the individual cryptographic mechanisms. This is due in part to the availability of an existing cryptographic library which simplified the implementation of the threshold secret sharing mechanisms in particular. Characteristics of this library which facilitated implementation are discussed in detail. Many of the complications in development are due to problems associated with combining cryptographic techniques and with implementing the underlying communication architecture which these techniques assume.

3.1 Background

After analyzing the requirements of the international treaty monitoring application, proactive DSA as described by Herzberg, Jakobsson, Jarecki, Krawczyk and Yung [HJJKY95], and Gennaro, Jarecki, Krawczyk and Rabin [GJKR96] was chosen as the algorithm to be implemented. While it is true that DSA is the United States government digital signature standard, DSA is still well accepted internationally. RSA is also very well accepted internationally. However, since RSA is frequently used for encryption, there are export limitations associated with it. Because of the inherent need to export the application and to provide source code as part of the exported product, these export issues are a definite drawback. In addition, while there has been significant work done in the area of proactive RSA particularly by Frankel, Gemmell, MacKenzie and Yung [FGMY97], there are some theoretical difficulties which have not yet been resolved making proactive RSA unsuitable for this application at this time as described below.

Initialization of key shares using proactive DSA is not particularly difficult cryptographically. This is because the moduli used in DSA are primes and are not required to be kept secret. The generation and use of the private parameter k is perhaps the most cryptographically complex part of the proactive DSA signature computation since k, in addition to the private key x, must remain secret and yet the value $g^{k^{-1}}$ must be computed by the shareholders in order to generate a signature. The mechanism described in the references is quite elegant and not difficult to implement.

Initialization of the key shares using proactive RSA on the other hand is a very difficult problem. The RSA modulus n is a composite of two large primes p and q. While n is a public value, p and q must remain secret. How do shareholders generate, in a distributed manner, a composite modulus n without knowing p and q and yet be assured that p and q are two large strong primes? At first glance, it would seem impossible. However, some advances recently have been made by Boneh and Franklin [BF97]. Due to their work, n can be generated in a distributed manner such that it is very probably a composite of two large primes. Still, several issues remain. First, there is no guarantee that p and q are strong primes. In addition, the technique described by Boneh and Franklin is not robust and for the international treaty monitoring application, robustness is required. Finally, the latest method for generating proactive RSA signatures is very complex and relies on patented mechanisms. However, a new technique described by Rabin [R97] promises to simplify proactive RSA signature generation greatly.

Based on this evaluation, proactive DSA is clearly the algorithm of choice for this application. However, it should be noted that there are mathematical restrictions placed on the relative values of t (the number of malicious adversaries the system can withstand) and n (the total number of possible shareholders in the system) which must be evaluated in conjunction with the actual political environment in which the system must function. In other words, it is possible that the political situation is such that the cryptographic restrictions on t and n cannot be met. This problem cannot even be addressed until the political properties of the application are understood. Clearly, these issues are outside the realm of this paper. Nonetheless, they are interesting to consider.

3.2 Existing Library

In coincidence with the availability of strong cryptographic methods, the variety of processing platforms warrants the need for portable and efficient implementations of strong algorithms. In fact, the plethora of potential applications of the various cryptographic algorithms motivates the use of software solutions that can readily adapt to changing requirements.

The software product supporting the implementation of proactive DSA, known as the Multiple-Precision Cryptographic Library (MPCL), is a family of multiple-precision arithmetic, hashing, and data authentication products based on modern asymmetric (public) key algorithms for producing digital signatures. These public key methods are unclassified and exportable to virtually all customers. The MPCL consists of a library of portable software modules, targeted for the host computer system that provide a standard interface to the MPCL family of products. The MPCL provides an interface to functions which generate and verify digital signatures as well as generating keys and cryptographic parameters in compliance with the Digital Signature Standard [F186]. The MPCL includes routines for performing modular multiple-precision mathematics which can be used for other cryptographic algorithms besides the DSA. The MPCL includes routines for hashing messages with the Secure Hash Algorithm (SHA-1) in compliance with the Secure Hash Standard [F180] and also with the MD5 hash algorithm [MvV97].

The MPCL was originally designed specifically for the Digital Signature Algorithm, but with an extensive suite of multiple-precision math and hash functions serves as a cryptographic developer's toolkit for use with other algorithms as well. The library is written in ANSI-C except for optional assembly language enhancements and is portable to multiple computing platforms. The primary feature of the MPCL is the efficiency afforded by the use of enhancements to modular math routines (e.g. pre-computed exponentiation tables) and the selective use of assembly language code. The flexibility of a software implementation allows new developments and discoveries of relevant cryptographic functions to be quickly realized.

The availability of source code is crucial to achieving a sound balance between efficiency, portability, and maintainability with a small staff. Efficiency improvements are generally realized in low-level routines that are often inaccessible in commercial cryptographic products. Portability without loss of efficiency normally requires subtle source code changes dependent on processor particulars. Interestingly, source code is also important because inspectability of cryptographic products is often necessary in international treaty monitoring environments.

The MPCL includes routines that build upon one another offering cryptographic functionality at various levels of abstraction. At the highest level, the MPCL provides the routines that users may call to execute initialization, key generation, signing and verification functions as well as functions related to proactive protocols. Access to lower level, building block routines, such as modular multiple-precision arithmetic is critical to the development of cryptographic algorithms. The MPCL also contains routines to perform hashing using the SHA and MD5 hashing algorithms. Selected routines from the library are listed in Table 1 with the highest level routines listed first.

Function	Purpose
KM_GenPQG	Key management function to generate DSA parameters
PDSA_Update*	Update shares of proactive DSA player's keys
PDSA_ShareRec	Share recovery routine for proactive DSA
PDSA_JointFeldmanRSS	Joint Feldman robust secret sharing routine
PDSA_JointUncondRSS	Joint unconditional robust secret sharing routine
PDSA_JointZeroSS	Joint zero knowledge secret sharing routine
DSA_Sign	Generate a digital signature
DSA_Verify	Verify a digital signature
LagrangeInterpolate	Lagrange interpolation routine
GenCommitment	Generation of Feldman or Pedersen commitments
VerifyCommitment	Verification of Feldman or Pedersen commitments
PolyAdd	Addition of two polynomials
PolyMult	Multiplication of two polynomials
PolyMultExp	Multiplication of two polynomials in the exponent
ModExp	Modular exponentiation function, A^B mod C
ModMult	Modular multiplication function, AB mod C
ModAdd	Modular addition function, $A+B$ mod C
ModInv	Modular inverse function, A^{-1} mod C
ModRNG	Modular random number generation function, RND(A) mod C
Div	Modular division and reduction function, A / B and A mod B
HashInit	Initialize the message hashing process
HashCont	Continue to hash a message
HashEnd	Complete the hashing process

Table 1. Selected routines provided in the MPCL package. Routines marked with an * are yet to be developed.

The method used to perform the multiple-precision modular arithmetic operations necessary to generate and verify DSA (and other) signatures is critical to the performance of the software. The parameters in DSA are integers that are much too large to be represented in ordinary processor registers by data types found in ordinary C compilers. Therefore, a scheme must be devised to represent these parameters and to perform the required arithmetic operations. Numbers are represented as a series of digits whose base is generally related to the processor word size. With a larger digit base, fewer operations necessary for a given multiple-precision operation. The MPCL easily accommodates changes in the digit base, thereby enhancing portability.

In the interest of execution speed, the time-critical operations of modular multiplication and modular exponentiation are performed using Montgomery's method for modular arithmetic [M85][DK90]. Using a paper by Kaliski [K95], the modular inverse function also takes advantage of Montgomery arithmetic. Montgomery multiplication can be written in the form of a convolution and is therefore especially well suited to digital signal processing chips. The features that make these processors effective is the existence of a multiply-and-accumulate (MAC) instruction and an accumulator with extra bits to hold the sum of the products in the convolution operations.

By far the most costly single operation in the DSA is the modular exponentiation which is used once during signing and twice during verification. The *fast exponentiation* algorithm implemented in the MPCL and discussed below in conjunction with an assembly language implementation of the Montgomery multiplication function results in a performance improvement of approximately 50 times.

The ability to improve the speed of exponentiation is critical to the performance of DSA with exponentiation being the single most computationally expensive operation for small to medium messages (for large messages, hashing can be the most expensive operation). In contrast to the DSA algorithm, a signature system based on the Rivest, Shamir, Adleman (RSA) algorithm involves modular exponentiations as well, but with a message dependent base. The modular exponentiation required in DSA involves a fixed base and modulus and a variable exponent. In other words, for each message, a new exponent will be generated, but the base and modulus will stay constant. This property allows the signer to pre-compute instantiations of the modular exponentiation for use in future computations. Because of the variability and size of the exponent, the pre-computations cannot be exhaustive. A clever method of pre-computing a series of powers of the base and utilizing them in an ordered fashion is described in [BGMW92] and as implemented in the MPCL offers a fivefold speed-up in modular exponentiation. Greater speed-up is possible as more exponentiations are pre-computed. Ordinarily, approximately 200 multiplications are necessary to perform the modular exponentiation of a 1024-bit base and modulus and a 160-bit exponent. The use of this *fast exponentiation* reduces the number of multiplications to approximately 40.

Any derivatives of the DSA algorithm can potentially take advantage of this modular exponentiation speed-up. In particular, threshold DSA signatures require the use of many exponentiations by all players in the system based on a constant base. Fixed-base exponentiation enhancements are also especially useful for most threshold secret sharing protocols.

The use of assembly language code generally improves the execution speed of programs written in a higher level language (including C). Although the portability of assembly language code is restricted to processors supporting the same instruction set, great advantages in the execution speed of digital signature algorithms can be gained by implementing a single routine in assembly language. In the MPCL, assembly language is used to improve the performance of the Montgomery multiplication function with a high payoff. The Montgomery multiplication function is the workhorse of the MPCL signing, verification, and threshold secret-sharing processes. This is because repeated multiplication is required by the time-critical modular exponentiation operation.

The MPCL includes assembly language implementations of the Montgomery multiplication function for the entire suite of the 8086 family of processors. Moreover, for some processors, including processors supporting the 80386 instruction set, the assembly language implementation of the Montgomery multiplication function allows doubling of the digit base used to represent multiple-precision integers. This feature alone results in a minimum performance increase of a factor of four. On these processors, assembly language code can take advantage of a 32-bit multiplication which results in a 64-bit result. On Sun workstations, a 64-bit integer data type (*long long*) can be used to implement 32-bit digits throughout the library.

Assembly language is an obvious divergence from an ANSI-C portable implementation. However, the performance improvement is substantial for the small amount of assembly language introduced into the MPCL. In addition, the assembly language enhancement is useful for fixed-based and non-fixed-base exponentiations (i.e. useful for both DSA and RSA). Ultimately, the use of assembly language is left as an option to the user of the library and a straight ANSI-C version is available.

Signing and verifying small (e.g. three bytes) messages allows establishment of a lower bound on the speed of signing/verifying. The performance results given in Table 2 involve the use of 1024-bits parameters. Every attempt was made on all platforms to avoid operating system overhead and interruptions.

Processing Platform	Fast Exp	ASM	Time to Sign (s)	Time to Verify (s)
Gateway P6-200	✔	✔	0.026	0.043
"		✔	0.093	0.172
"	✔		0.187	0.295
"			0.928	1.440
50 Mhz Sparcstation 20	✔		0.164	0.314
"			0.792	1.565

Table 2. Performance results of signing and verifying on selected computing platforms.

The enhancement that offers the biggest payoff is the assembly language implementation of the Montgomery multiplication function, at least when the digit base can be doubled. However, there is never a reason not to use the *fast exponentiation* for DSA and other appropriate cryptographic mechanisms except in memory limited systems. In these systems, the software can be configured to turn off this option.

3.3 Algorithm Description

The following is a description of the DSS-Thresh-Sig-3 protocol used to jointly generate a DSA signature. The protocol is 1/3 resilient (less than 1/3 of the participants can be malicious). The complete set of definitions, tools and the description of the model are found in [GJKR96]. Our presentation is only intended to aid the reader.

3.3.1 Notation

p - prime number

q - prime number, s.t. $p = mq + 1$, where m is a small integer.

g - an element of Z_p of order q, i.e., $g^q = 1 \bmod p$.

h - an element of Z_p of order q, and $h = g^d, 1 < d < q$.

n - the number of shareholders, $n \in Z_q$.

t - the number of malicious players the signature scheme can tolerate, $t < n$.

(1) $(s_1, \ldots, s_n) \xleftarrow{(t,n)} s \bmod q$ denotes a sharing of a secret s using Shamir's Threshold Secret Sharing scheme.

(2) $P^{(i)}(\alpha) = P_t^{(i)}\alpha^t + P_{t-1}^{(i)}\alpha^{t-1} \ldots + P_1^{(i)}\alpha + P_0^{(i)} \bmod q$ - a polynomial of degree t with coefficients of random numbers in Z_q, generated by player i. $P_0^{(i)}$ is player i's secret.

(3) $p_j^{(i)} = P^{(i)}(j)$ - player j's share of player i's secret.

(4) $P(\alpha) = P_t\,\alpha^t + P_{t-1}\alpha^{t-1} \ldots + P_1\,\alpha + P_0 = \sum_{i=1}^{n} P^{(i)}(\alpha) \bmod q$ - the global polynomial (unknown to any t of the players) that is used to share the (equally unknown) group secret P_0. Obviously

(5) $P_0 = \sum_{i=1}^{n} P_0^{(i)} \bmod q$.

(6) $p = P(0) = P_0$ - the secret of which every player has a share.

(7) $p_i = \sum_{j=1}^{n} p_i^{(j)} = \sum_{j=1}^{n} P^{(j)}(i)$ - player i's share of the secret p. Using notation (1), $(p_1,...,p_n) \xleftarrow{(t,n)} p$.

(8) $g^{P_t^{(i)}}, g^{P_{t-1}^{(i)}}, ..., g^{P_1^{(i)}}, g^{P_0^{(i)}} \bmod p$ - player i's Feldman commitments to his $P^{(i)}(\alpha)$ polynomial.

(9) $g^{P_t^{(i)}} h^{F_t^{(i)}}, g^{P_{t-1}^{(i)}} h^{F_{t-1}^{(i)}}, ..., g^{P_1^{(i)}} h^{F_1^{(i)}}, g^{P_0^{(i)}} h^{F_0^{(i)}}$ - player i's Pedersen commitments to his $P^{(i)}(\alpha)$ polynomial, where $F_l^{(i)}, l = 0..t$ are the coefficients of an auxiliary polynomial $F^{(i)}(\alpha)$ generated by player i.

3.3.2 Tools

The following high-level tools are required to implement proactive DSA:

Joint Feldman Random Secret Sharing (n, t, s) - n players end up sharing a random secret s (unknown to any one of them) by having each player generate a t-degree polynomial with random coefficients in Z_q and distribute shares of his polynomial to the other players. The correctness of shares is verified using Feldman's Verifiable Secret Sharing scheme.

- Player i generates the t-degree polynomial $S^{(i)}(\alpha)$.
- Player i evaluates his polynomial at $j = 1,...,n$ and sends $s_j^{(i)} = S^{(i)}(j)$ to player j over a private point-to-point channel.
- Player i calculates the Feldman commitments to the polynomial coefficients as in (8) and broadcasts them.
- Player i verifies that player j's commitments to him were correct by evaluating j's polynomial at point i in the exponent:

  (10) $g^{S^{(j)}(i)} \stackrel{?}{=} (g^{S_t^{(j)}})^{i^t} (g^{S_{t-1}^{(j)}})^{i^{t-1}} ... (g^{S_1^{(j)}})^{i^1} (g^{S_0^{(j)}}) \bmod p$.

  Players for which (10) does not verify are eliminated and the shares distributed by them ignored.
- Player i's share of s, s_i, is calculated by summing up the shares received from the good players, as in (7).

In the end $\{s_i | i \in GoodPlayers\} \xleftarrow{(t,n)} s$.

Joint-Unconditionally-Secure-Random-Secret-Sharing (n, t, s) - n players end up sharing a random secret s (unknown to any one of them) by having each player generate a t-degree polynomial with random coefficients in Z_q and distribute shares of his polynomial to the other players. The correctness of shares is verified using Pedersen's Unconditionally Secure Verifiable Secret Sharing scheme.

- Player i generates the t-degree polynomial $S^{(i)}(\alpha)$ and an auxiliary t-degree polynomial $F^{(i)}(\alpha)$.
- Player i evaluates his polynomials at $j = 1,...,n$ and sends $s_j^{(i)} = S^{(i)}(j)$ and $f_j^{(i)} = F^{(i)}(j)$ to player j over a private point-to-point channel.
- Player i calculates the Pedersen commitments to the polynomial coefficients as in (9) and broadcasts them.
- Player i verifies that player j's commitments to him were correct by evaluating the sum $S^{(j)}(i) + cF^{(j)}(i)$ in the exponent:

  (11) $g^{S^{(j)}(i)} h^{F^{(j)}(i)} \stackrel{?}{=} (g^{S_t^{(j)}} h^{F_t^{(j)}})^{i^t} (g^{S_{t-1}^{(j)}} h^{F_{t-1}^{(j)}})^{i^{t-1}} ... (g^{S_1^{(j)}} h^{F_1^{(j)}})^{i^1} (g^{S_0^{(j)}} h^{F_0^{(j)}}) \bmod p$.
- Players for which (11) does not verify are eliminated and the shares distributed by them ignored.

- Player i's share of s, s_i, is calculated by summing up the shares received from the good players, as in (7).

Joint Zero Verifiable Secret Sharing (n, t, A) - n players end up sharing a "secret" whose value is zero by having each player generate a t-degree polynomial $A^{(i)}(\alpha)$ with constant term zero and random coefficients in Z_q and distribute shares of this polynomial to the other players. The correctness of shares is verified using Feldman's Verifiable Secret Sharing scheme.

- Player i generates the t-degree polynomial $A^{(i)}(\alpha)$.
- Player i evaluates his polynomial at $j = 1,...,n$ and sends $a_j^{(i)} = A^{(i)}(j)$ to player j over a private point-to-point channel.
- Player i calculates the Feldman commitments to the polynomial coefficients as in (8) and broadcasts them.
- Player i verifies that player j's commitments to him were correct by evaluating j's polynomial at point i in the exponent:

(12) $$g^{a_0^{(j)}} \stackrel{?}{=} 1 \text{ and}$$
$$g^{S^{(j)}(i)} \stackrel{?}{=} (g^{S_t^{(j)}})^{i^t} (g^{S_{t-1}^{(j)}})^{i^{t-1}} ... (g^{S_1^{(j)}})^{i^1} \bmod p$$

  Players for which (12) does not verify are eliminated and the shares distributed by them ignored.
- Player i's share of 0 is calculated by summing up the shares received from the good players as in (7).

Multiplying Two Secrets(n,t,a,b) - n players are holding shares of two secrets, a and b, shared with t-degree polynomials $A(\alpha)$ and $B(\alpha)$: $(a_1,...,a_n) \xleftrightarrow{(t,n)} a$ and $(b_1,...,b_n) \xleftrightarrow{(t,n)} b$. How do they calculate $c = ab \bmod q$ without reconstructing a or b?

- Player i calculates product $c_i = a_i b_i = A(i)B(i) \bmod q$. He therefore obtains a share of the product polynomial $C(\alpha) = A(\alpha)B(\alpha)$ of degree $2t$.
  Note that since $a = A(0)$ and $b = B(0)$, $ab = C(0)$.
- Player i broadcasts c_i.
- Any $2t+1$ good shares of $C(\alpha)$ can be used to reconstruct $ab = C(0)$.

To hide the $a_i b_i$ product, a constant term zero polynomial, $D(\alpha)$ of degree $2t$ can be used (players execute Joint-Zero-VSS(n,$2t$,D)), such that $C(\alpha) = A(\alpha)B(\alpha) + D(\alpha) \bmod q$ and $C(0) = A(0)B(0) \bmod q$, or $c = ab \bmod q$. In this case player i would calculate $c_i = a_i b_i + d_i \bmod q$, and any $2t+1$ good c shares could be used to compute $c = C(0)$. We will refer to this procedure as Multiply-Two-Secrets(n,t,a,b,d,c).

Verifying that a player's c_i is really $a_i b_i$ could be done using error correcting codes. That allows for $\frac{n-1}{4}$ bad shares. The following section describes how this can be achieved using Verifiable Secret Sharing and tolerate $\frac{n-1}{3}$ bad shares as a result.

DSS-Thresh-Sig-3-Verification(n,t,a,k,b,v)

Secret a has been shared with Joint-Feldman-RSS using t -degree polynomials $A^{(i)}(\alpha), i = 1..n$.
Constant term zero $2t$ -degree polynomials $B^{(i)}(\alpha), i = 1..n$ have been used to share zero.
Secret k has been shared with Joint-Uncond-Secure-RSS using polynomials $K^{(i)}(\alpha), i = 1..n$ (notice that $g^{k_i}, i = 1,\ldots,n$ are not publicly known).

Secret information held by player i :
$$\begin{cases} a_i = \sum_{j=1}^{n} A^{(j)}(i) \bmod q \\ K^{(j)}(i), 1 \le j \le n \\ b_i = \sum_{j=1}^{n} B^{(j)}(i) \bmod q \end{cases}$$

Public information:
$$\begin{cases} g^{A_l^{(i)}} \bmod p,\ g^{A_l} = \prod_{i=1}^{n} g^{A_l^{(im)}} \bmod p,\ 0 \le l \le t,\ g^{a_i} = \prod_{l=0}^{t} (g^{A_l})^{i^l} \bmod p,\ 1 \le i \le n \\ g^{B_l^{(i)}} \bmod p,\ g^{B_l} = \prod_{i=1}^{n} g^{B_l^{(i)}} \bmod p,\ 0 \le l \le t,\ g^{b_i} = \prod_{l=0}^{t} (g^{B_l})^{i^l} \bmod p,\ 1 \le i \le n \end{cases}$$

How do players get to own good shares of the polynomial $V(\alpha) = A(\alpha)K(\alpha) + B(\alpha) \bmod q$?
Hint: Have player i commit to $W^{(i)}(\alpha) = A(\alpha)K^{(i)}(\alpha) \bmod q$.

- Player i calculates the additional commitments to his $W^{(i)}(\alpha)$ polynomial (we will call these the DSS-Thresh-Sig-3 commitments), $g^{W_l^{(i)}} = \prod_{j_1+j_2=l} (g^{A_{j2}})^{K_{j2}^{ii)}} \bmod p, 0 \le l \le 2t$, and broadcasts them.
- Player i verifies player j's commitment to $A(\alpha)K^{(j)}(\alpha)$ at i :
  (13) $(g^{a_i})^{K^{(j)}(i)} =^{?} (g^{W_{2t}^{(j)}})^{i^l} (g^{W_{2t-1}^{(j)}})^{i^{l-1}} \ldots (g^{W_0^{(j)}}) \bmod p$
  Players for which (13) does not verify are excluded from *GoodPlayers* and the $K^{(j)}(i)$ shares distributed by them ignored.
- Player i calculates $k_i = \sum_{j \in GoodPlayers} K^{(j)}(i) \bmod q$, $v_i = a_i k_i + b_i \bmod q$ (use Multiply-Two-Secrets (n,t,a,k)) and broadcasts v_i .
- Player i verifies player j's v_j :
  $g^{B(j)} = (g^{B_{2t}})^{j^{2t}} (g^{B_{2t-1}})^{j^{2t-1}} \ldots (g^{B_1})^{j} (g^{B_0}) \bmod p$
  $g^{W_l} = \prod_{m \in GoodPlayers} g^{W_l^{(m)}} \bmod p, \quad 0 \le l \le 2t$
  $g^{W(j)} = (g^{W_{l2t}})^{j^{2t}} (g^{W_{2t-1}})^{j^{2t-1}} \ldots (g^{W_1})^{j} (g^{W_0}) \bmod p$
  (14) $g^{B(j)} g^{W(j)} \stackrel{?}{=} g^{v_i} \bmod p$

  Players for which (14) does not verify are excluded from *GoodPlayers* and their v_j shares ignored.

3.3.3 DSS Signature Generation - Protocol DSS-Thresh-Sig-3

The following protocol is directly from [GJKR96] with only minor clarifications which we found helpful as implementers.

1. **Generate k**
   The players generate a secret value k, uniformly distributed in Z_q, by running Joint-Uncond-Secure-RSS(n,t,k). Notice that this generates $(k_1,..k_n) \xleftarrow{(t,n)} k \bmod q$.

   Secret information of P_i: a share k_i of k

2. **Generate random polynomials with constant term 0**
   Execute two instances of Joint-Zero-VSS with polynomials of degree $2t$: Joint-Zero-VSS($n,2t,B$), Joint-Zero-VSS($n,2t,C$). Denote the shares created in these protocols as $\{b_i\}_{i\in\{1..n\}}$ and $\{c_i\}_{i\in\{1..n\}}$.

   Secret information of P_i: shares b_i, c_i

   Public information: $g^0 = 1, g^{b_i}, g^0 = 1, g^{c_i}, 1 \le i \le n$

3. **Generate $r = g^{k^{-1}} \bmod p \bmod q$** (Use Multiply-Two-Secrets(n,t,a,k,b,v) + DSS-Thresh-Sig-3-Verification(n,t,a,k,b,v))
   (a) Generate a random value a, uniformly distributed in Z_q^*, using Joint-Feldman-RSS(n,t,a). Calculate and broadcast the DSS-Thresh-Sig-3 commitments.

   Secret information of P_i: a share a_i of a

   Public information: $g^a, g^{a_i}, 1 \le i \le n$

   $$g^{W_l^{(i)}} = \prod_{j_1+j_2=l} (g^{A_{j_1}})^{K_{j_2}^{(i)}} \bmod p, 0 \le l \le 2t, 1 \le i \le n$$

   (b) Player P_i broadcasts $v_i = k_i a_i + b_i \bmod q$. If P_i doesn't broadcast a value set v_i to null.
   (c) Player P_i verifies all other players' v shares with DSS_Thresh-Sig-3-Verification(n,t,a,k,b)

   Public information: $v_1,...,v_n$ where for at least $n-t$ values j it holds that $v_j = k_j a_j + b_j \bmod q$

   (d) Player P_i computes locally

   $\mu = Interpolate(v_1,...,v_n) \bmod q \quad [= ka \bmod q]$

   $\mu^{-1} \bmod q \quad [= k^{-1}a^{-1} \bmod q]$

   $r = (g^a)^{\mu^{-1}} \bmod p \bmod q \quad [= g^{k^{-1}} \bmod p \bmod q]$

   The DSS-Thresh-Sig-3 commitments for step (4) can now be generated.
   Note: Even though the above computations are local, as they are done on public information we can assume that r is public:

   Public information: r

   $$g^{Z_l^{(i)}} = \prod_{j_1+j_2=l} (g^{X_{j_1}})^{K_{j_2}^{(i)}}, 0 \le l \le 2t, 1 \le i \le n,$$

   with $X'(\alpha) = rX(\alpha) + m$

4. **Generate $s = k(m + xr) \bmod q$** (Use Multiply-Two-Secrets($n,t,(m+rx),k,c,s$) + DSS-Thresh-Sig-3-Verification($n,t,(m+rx),k,c,s$))

   (a) Player P_i broadcasts $s_i = k_i(m + x_i r) + c_i \bmod q$.
   (b) Player P_i verifies all other players' shares with DSS-Thresh-Sig-3-Verification($n,t,(m+rx),k,c,s$)

Public information: $s_1,\ldots,s_n$ where for at least $n-t$ values j it holds that $s_j = k_j(m + x_j r) + c_j \bmod q$

(c) Set $s = Interpolate(s_1,\ldots,s_n)$.

5. **Output the pair** ***(r,s)***

3.4 Protocol

Once the individual cryptographic mechanisms are developed, they must be combined to produce the proactive protocol. The protocol is designed as a series of state machines. Some complex states expand into individual state machines. In the following diagrams, these complex states will be noted by shading. Not all states will be discussed in depth in this paper. At a high level, the protocol is represented as:

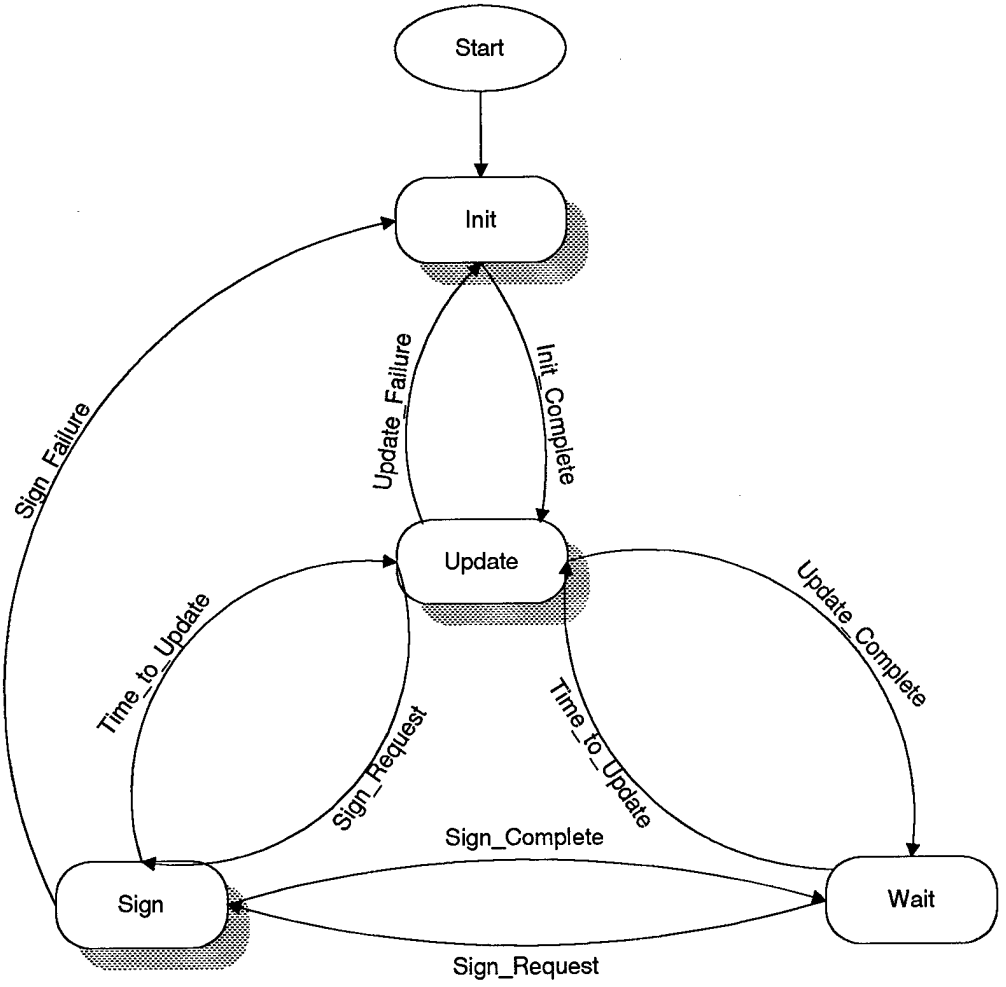

Figure 3. Proactive DSA High-Level State Diagram

The system is initialized with the necessary parameters and the initial key shares are generated. Once initialization has been performed, the system loops indefinitely between the Update, Sign and Wait states

unless it becomes apparent that too many shareholders have been compromised. In this case, the Init state is re-entered.

The following diagram represents the Update state.

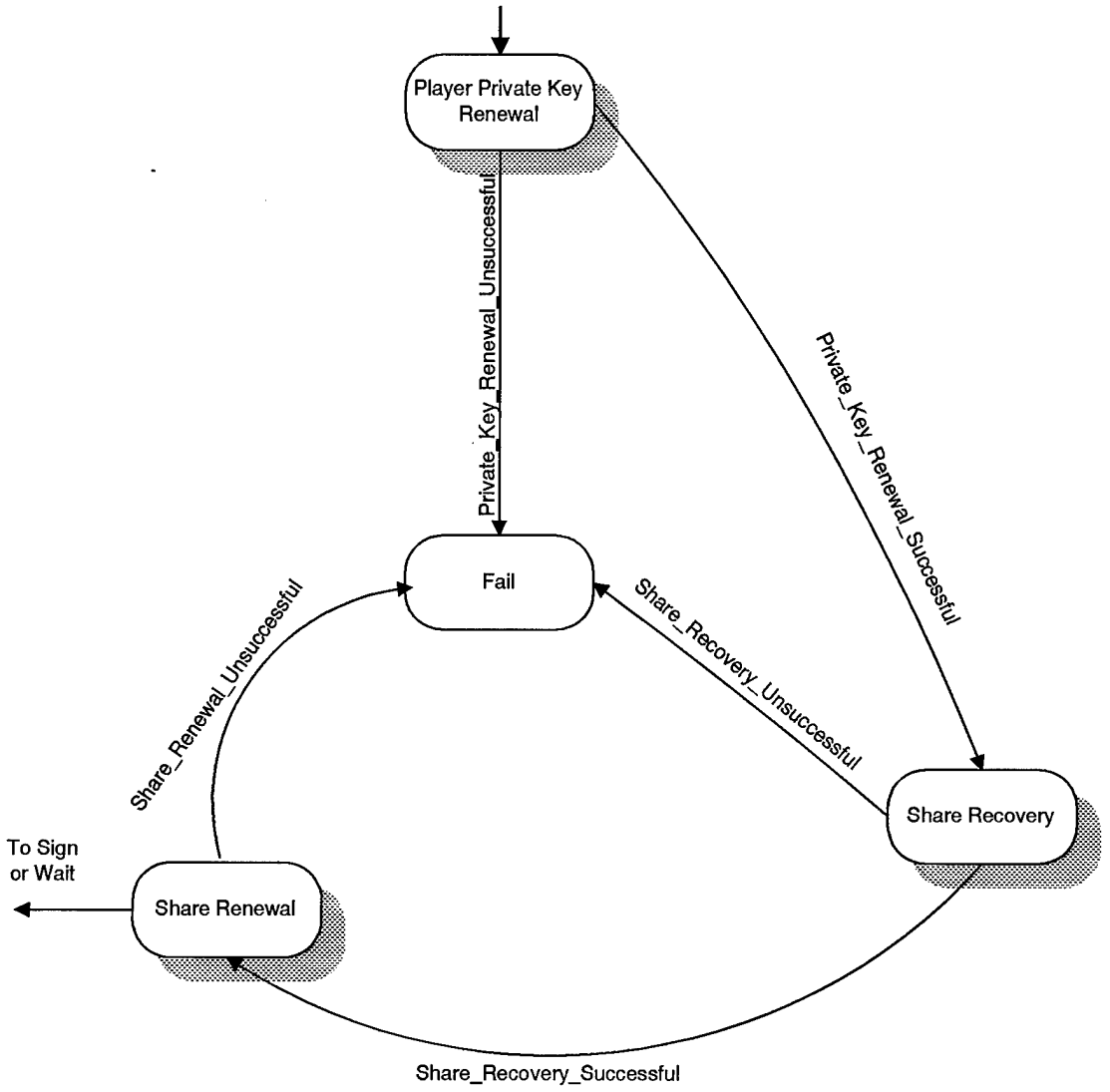

Figure 4. Proactive DSA Update State

In the Update state, player (or shareholder) individual keys are updated, key shares are recovered as needed and updated as described by Jarecki [J96]. In this state, shareholders which have been corrupted can re-enter the system. Corrupted shareholders may only re-enter the system during an Update. A Fail state is possible which would cause the system to re-enter the Init state.

Finally, the following diagram defines the Sign state.

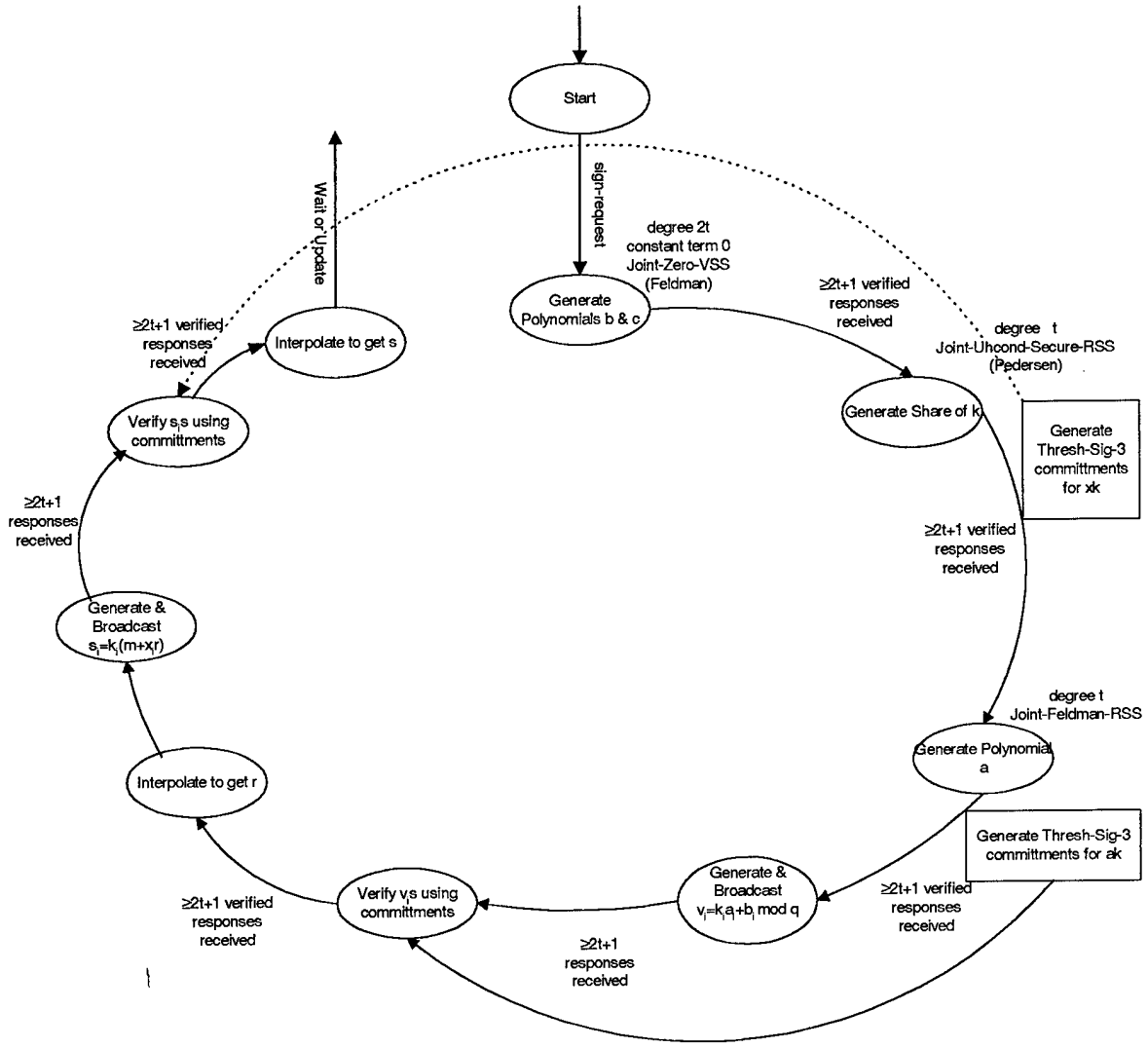

Figure 5. Proactive DSA Sign State

The boxes indicate where the DSS-Thresh-Sig-3 commitments are generated. The dotted lines indicates where they are actually used.

Although not shown, a Fail state is possible causing the system to re-enter the Init state. Notice that at each transition in which there is a required number of responses, the number of responses is at least $2t + 1$. This is true even when the polynomial being generated is of degree t. The reason for this requirement is that in the end at least $2t + 1$ verified responses will be required. There is no need to progress if less than $2t + 1$ responses are received. The number of verified responses required and the number of adversaries the system can withstand implies that $n \geq 3t + 1$ (i.e., $2t + 1 + t$). This diagram shows verification occurring at individual transitions. In fact, as long as all messages are saved, the system can be configured to verify only at the end. However, as we shall see, the number of messages can become quite large. Still, this may be a far more efficient implementation if it is likely that verifiability in and of itself is enough to deter adversaries. Notice also that this diagram implies that the shareholders *agree* about the current status of the system (e.g., the state, the good shareholders, etc.). There must be a mechanism for this agreement.

Implementation of proactive DSA as described in the references assumes an underlying communication architecture. Specifically:

- pair-wise private channels must exist, and
- a reliable broadcast mechanism must be provided.

Pair-wise private channels are easy to implement. We have already indicated that each shareholder has his own key pair in addition to his share of the system private key. Using Diffie-Hellman key exchange, pair-wise session keys can be generated and used as needed. It is important to note that the encryption mechanism must be semantically secure. Introduction of encryption may or may not lead to export issues which would have to be resolved prior to installation.

The reliable broadcast mechanism is a much more difficult problem. We are implementing a variation of Byzantine Agreement [FM88] as described by Gassko and Gemmell in [GG97]. Their technique makes use of public key cryptography to provide reliable broadcast in the presence of up to $\lfloor n/2 \rfloor$ adversaries. Once again, each shareholder will use his individual private key to authenticate messages. In addition, it is this mechanism that allows shareholders to agree on the current state of the system.

The mechanism we use is as follows:

1. For any message a shareholder originates (as opposed to merely retransmits), the shareholder signs the message with his private key and sends it to all other shareholders.
2. Each shareholder receiving the message retransmits it to all other shareholders unchanged.
3. Each shareholder now compares all the messages. If they are the same, then the original message is accepted and no signature verifications need take place. On the other hand, if messages differ, verification must occur. If two different signatures verify, then the other shareholders assume that the shareholder originating the message has been corrupted. If less than $\lfloor n/2 \rfloor+1$ signatures verify, then the shareholder originating the message is assumed to be corrupted. In all other cases, the message is assumed to be the one which verified.

Notice that the number of communications for each message is about n^2. If the system is configured to perform verification only at the end of the signature process, all of these messages must be saved for use during this process.

Finally, we have opted to implement synchronous as opposed to asynchronous communications. There is a clear performance penalty that comes with this choice. However, the cryptographic mechanisms and communication architecture present problems which are challenging enough to a small development team without introducing the complexity associated with asynchronous communications. The system is simply easier to manage if the underlying communications are synchronous. A global clock or global time server is not appropriate for this application. To synchronize shareholders, we propose to use relative time with system parameters which represent maximum delay through the network and maximum clock drift [HT93] of all clocks in the system.

3.5 Status

We believe that we now have a complete design which can be implemented. All of the cryptographic mechanisms have been developed. Currently, we are implementing the underlying communication network on a series of Windows NT machines. In addition, we are implementing the processes based on the state machines depicted in the diagrams in section 3.4 and expect to have a working prototype by the second quarter of 1998.

4. Acknowledgments

The authors would like to thank Yair Frankel for introducing them to the world of proactive cryptography, Ernie Brickell and Kevin McCurley for technical guidance in implementing the fast exponentiation algorithm, Pete Gemmell, Irina Gassko, and Amy Johnston for their patience in answering general cryptographic questions, Sorin Istrail for his direction on general mathematics questions, Marjorie Jimenez and Karen Shanklin for project support, and Tal Rabin for her encouragement in implementing proactive DSA.

References

[BF97] D. Boneh and M. Franklin. Efficient Generation of Shared RSA Keys, *Advances in Cryptology - CRYPTO '97*.

[BGMW92] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast Exponentiation with Precomputation, *Advances in Cryptology - EUROCRYPT '92*.

[DF89] Y. Desmedt and Y. Frankel. Threshold Cryptosystems, *Advances in Cryptology - CRYPTO '89*.

[DK90] S. R. Dusse and B. S. Kaliski Jr. A Cryptographic Library for the Motorola DSP56000, *Advances in Cryptology - EUROCRYPT '90*.

[FM88] P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine Agreement, *Proceedings 20th ACM Symposium on Theory of Computing*, 1988.

[FGMY97] Y. Frankel, P. Gemmell, P. MacKenzie, and M. Yung. Optimal-Resilient Proactive Public-Key Cryptosystems, *Proceedings of the Symposium on Foundations of Computer Science*, 1997.

[GG97] I. Gassko and P. Gemmell. Implementation Issues for Threshold Protocols, manuscript, July 1997.

[GJKR96] R. Gennaro, S. Jarecki, H. Krawczyki, and Tal Rabin. Robust Threshold DSS Signatures, *Advances in Cryptology - EUROCRYPT '96*.

[HT93] V. Hadzilacos, S. Toueg. Fault-Tolerant Broadcasts and Related Problems. *Distributed Systems, Second Edition*, Addison-Wesley, 1993.

[HJJKY97] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Public Key and Signature Systems, *The Fourth ACM Symposium on Comp. and Comm. Security*, April 1997.

[HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M.Yung. Proactive Secret Sharing or How to Cope with Perpetual Leakage, *Advances in Cryptology - CRYPTO '95*.

[J96] S. Jarecki. Proactive Secret Sharing and Public Key Cryptography, *Master Thesis*, MIT, 1996.

[K95] B. S. Kaliski Jr. The Montgomery Inverse and its Applications, *IEEE Transactions on Computers*, Vol. 44, No. 8, August, 1995.

[MvV97] A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography, CRC Press, 1997.

[M85] P. L. Montgomery. Modular Multiplication without Trial Division, *Mathematics of Computation*, 44(170):519-521, 1985.

[R97] T. Rabin. Proactive RSA, *CRYPTO '97 Rump Session*.

[S79] A. Shamir. How to Share a Secret, *Communications of the ACM*, Vol. 22, 1979.

[FI186] FIPS 186, Digital Signature Standard, Federal Information Processing Standards Publication 186, U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, Virginia, 1994.

[FI180] FIPS 180-1, Secure Hash Standard, Federal Information Processing Standards Publication 186, U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, Virginia, April 17, 1995 (supersedes FIPS PUB 180).

Report Number (14) SAND--97-2939C
CONF-980534--

Publ. Date (11) 19980503
Sponsor Code (18) DOE/DP, XF
UC Category (19) UC-700, DOE/ER

DOE