"Pseudorandom Intermixing": A Tool for Shared Cryptography

Yair Frankel[1], Philip MacKenzie[2], and Moti Yung[3]

[1] CertCo, N.Y., NY, yfrankel@cryptographers.com, yfrankel@cs.columbia.edu
[2] Bell Laboratories, Murray Hill, NJ, philmac@research.bell-labs.com
[3] CertCo, N.Y., NY, moti@certco.com, moti@cs.columbia.edu

Abstract. Designing distributed cryptographic protocols that combine correctness, security, efficiency and practical constraints can be very difficult. Here, we suggest a new modular tool that we call *"pseudorandom intermixing"* which allows parties (or architectural components, such as hardware devices) sharing pseudorandomness to mix extra correlated pseudorandom information inside their computational results. We show how the pseudorandom intermixing may ease the design, increase efficiency and allow more refined control of cryptographic protocols for several important tasks, while maintaining "provable security." It can even turn a "heuristic protocol" into a "provably secure" one.

We concentrate on the area of "distributed public key systems," which has been a very active area of research in the last decade, and for which there is a great interest in practical implementations of protocols. Among other things, we demonstrate the first "fault-free non-interactive" proactive maintenance protocol for RSA, which involves a single broadcast round to perform an update, if parties do not behave maliciously. We also demonstrate how to interlace access control within the messaging of proactive RSA; this assures elimination of corrupted entities.

1 Introduction

Cryptographic protocol development is often a challenge for it requires careful design to assure both security and efficiency. Therefore, general design mechanisms which can be incorporated into protocols in a modular fashion and provide easier proofs of security are highly desired.

In this work we develop a tool which we call *pseudorandom intermixing*. In a nutshell, it is a notion that assures *entangling* of pseudorandom information shared between parties with *the cryptographic (algebraic) computational results themselves!* Namely, the pseudorandom intermixings change the local outcomes of computations by adding pseudorandom elements to the computation, in a way much similar to computing with a randomized key schedule.

As a result of pseudorandom intermixing, the local outputs appear random, which makes it easier to simulate the protocol and prove its security. However, since the outputs are not random, but rather t-wise random, and the intermixing is deterministically generated (pseudorandom), they allow for handshaking (assurance of collaboration) amongst all the active players.

H. Imai, Y. Zheng (Eds.): PKC 2000, LNCS 1751, pp. 306–325, 2000.

A major area of applications of pseudorandom intermixing is distributed cryptosystems, an important field which enables distributed trust, added security, increased availability, and avoidance of single points of failure (for surveys on the subject see [D92,FY98]). In particular we concentrate on distributed systems based on RSA [RSA], which is perhaps the most widely used public-key function. Such distributed RSA systems have been shown possible in various security/adversarial models [B88,F89,DF91,DDFY,FGY,GJKR96,FGMY,FGMY2,R]. The cryptographic objective of a distributed threshold RSA function-sharing system is to distribute the RSA signing capability so that any t or more entities can sign a message, yet an adversary that compromises at most $t-1$ entities can not sign.[1]

1.1 The Basic Issues

In addition to considering enhanced security and efficiency of cryptographic protocols, we also consider enhanced control aspects which allow the parties better tools to assure policy adherence.

In fact, many of the previous threshold cryptosystems missed vital control aspects necessary for a sound and secure deployment when taking into consideration the complete systems perspective and not just the purely mathematical aspects of distributed function application. Here we resolve many of the open issues of a secure system design for threshold cryptosystems by adding control into the operation without adding much cost. In fact, we interlace access control management into the design. To demonstrate the need, let us suppose that an entity has been found to be corrupted (for example, an employee who is quitting or being fired, an entity who lost a key due to a virus attack, etc.). Then, in several threshold cryptosystems, when the adversary has the outputs of $t-1$ good parties then the adversary can make the signature. But, this is not what the policy should allow! Rather, it should require a t quorum of good components. It should further require that when one is identified as corrupt then that entity should have no capabilities whatsoever. Indeed, proactive system [OY] resolve the problem of cleaning up the system and ignoring components known to be corrupt. However, they require all entities to be present and participate in a costly update protocol in order to effectively remove a party. A better solution would be to require the working parties to perform a handshake in order to know and validate whom they are working with. Though completely non-interactive protocols (e.g., [F89,DDFY,Sho]) may be efficient and mathematically interesting, there are situations where they may have reduced security when corrupted players are identified and when the policy requires to ignore them. (In addition, the system audit mechanisms may require anyway to know which elements are active). Pseudorandom intermixing interlaces t-wise access control to allow the protocol to be non-interactive except for the announcement of the active players

[1] This is the same protection as in secret sharing [Bl,Sh], but the signing operation may be repeated as long as the key is valid.

(entities) within a round or a period and yet to enable access control based on known dishonest/unavailable parties.

There are various other operational aspects. One important aspect of systems is that the system must have minimal availability and security requirements yet require minimal resources. Let us give an example based on the proactive RSA ramp [2] scheme [R]. This system in its simple form allows for a reduction in the available players to perform future signatures when an entity becomes inactive yet uncorrupted. Moreover, to turn the RSA ramp scheme back into an RSA threshold scheme, we require (for a scheme with a total of l entities) either private memory of size $O(l^3)$ or interaction of all players after each signature via a proactive update, making it into an l-out-of-l system (visualize the availability requirement change when l=100 and t=3). Both measures result in extremely cumbersome systems from any practical perspective and the simplicity of the simplified approach is lost. Another example is the non-interactive robustness assurance of [GJKR96] with check vectors. This method requires incorporating additional security requirements on parties which normally are treated as unsafe (e.g., the combiner is required to have secret keys–this is against the original model where the combiner is merely an abstraction). Such a single point of failure is dangerous–it requires a unique combiner to be available to assure robustness of operation.

The above examples, show that many designs which are legitimate mathematical exercises resulting in new ideas which may even optimize a single parameter (and we do not object to such investigations per se), may nevertheless be undesirable in actual systems. The balance of availability, security, efficiency and flexible control is essential in the design of practical systems.

1.2 Our Results

Our specific results on distributed cryptosystems include:

Security By incorporating pseudorandom intermixing, we are able to convert the simple yet heuristic threshold RSA protocol of [DF91] into a provably secure protocol. Pseudorandom intermixing is able to achieve this by converting public values which are not known to be simulatable into random-looking values that are easy to simulate.

Efficiency By incorporating pseudorandom intermixing, we are able to reduce the communication requirements of the secure threshold RSA protocol of [FGMY2] and reduce the communication of robustness/checking information in the robust threshold RSA of [FGMY2]. Moreover, we show how the shared pseudorandomness enables elimination of interaction, turning interactive protocols into "fault-free non-interactive" ones (a notion of efficient protocols put forth in [F]). In particular, we show how to use pseudorandom intermixing to eliminate interaction in the distributed RSA signing protocol of [FGMY2] and give the first non-interactive proactive update.

[2] See [BM] for a reference to ramp schemes; essentially it allows for reduced threshold throughout the system's operation.

Access Control Pseudorandom intermixing allows easy access control management, where parties can be included (excluded) from a designated "working set" by including (excluding) the parties' shared pseudorandomness from the computation. As a result, the correctness of the final result in a distributed computation implies the correct working set is involved.

Isolation Pseudorandom intermixing individualizes each computation, by effectively changing the key based on the common message. This can help in foiling side-channel cryptanalysis (based on timing or power consumption leakage).

We feel that the practice of cryptographic systems needs generic tools that can, in a modular fashion, enhance various properties of numerous protocols and working environments (as we mention above). Pseudorandom intermixing is such a tool and is based on mixing public and private key cryptography in a new way. It seems to us that perhaps it may provoke more thinking about new cryptographic tools with wide modular applicability.

We comment that the use of shared randomness in distributed protocols and how it enables certain designs was recently presented (in an independent work) [GI99] where shared randomness is treated as a "compressed resource." They used the resources for general multi-party computation and proactive secret sharing. Our work has a bit different flavor: we show how the resource can be integrated into "existing protocols" in a modular fashion, thus showing the "incremental nature" of the resource, also our area of application is the more pragmatic "distributed cryptosystems." In addition, we pseudorandomly derive the correlated randomness from common inputs, and thus effectively "individualize" each computation, which may help against side-channel attacks (e.g. timing attacks). The general idea of building basic tasks from shared randomness was first put forth in a generic theoretical context in [B97].

Organization: We discuss some basic tools we use in Section 2, and we provide an informal description of pseudorandom intermixing in Section 3. We then present numerous applications of pseudorandom intermixing to threshold cryptography in Section 4. Finally, in Section 5 we give applications to fault tolerant protocols (robust and proactive ones).

2 Background

Here we describe the basic cryptographic functions and components that we use in the paper.

The RSA algorithm: The RSA key generator produces two large random primes p_1 and p_2, and computes a composite $n = p_1 \cdot p_2$ and the Euler totient function $\phi(n) = (p_1 - 1)(p_2 - 1)$. The generator chooses a public key $\langle e, n \rangle$, where $\gcd(e, \phi(n)) = 1$, and private key d, such that $ed \equiv 1 \bmod \phi(n)$. The one-way (hard) direction of RSA (used in decryption or signature of a message m) is $S_m \equiv m^d \bmod n$, whereas the public (easy) direction (used in encryptions and verification of signature) is the inverse function $z^e \bmod n$ for a message

z. Typically for signatures, the message m to be raised to the power d is a cryptographic hash of the real message M.

Discrete logarithm: Let P be a prime and g a generator of a subgroup of Z_P of large order. The function $f(x) = g^x \bmod P$ is one-way (finding x is called the discrete log problem). The Diffie-Hellman key exchange [DH] can be built on it. We can similarly define discrete logarithms over composites.

Pseudorandom functions: With a pseudorandom function family, given a random member of the family the result at any chosen input point looks random, and sampling it polynomially many times cannot distinguish it from a truly random function. We often denote a pseudorandom function family as PRF and an element from PRF indexed by k as PRF_k. Formal definitions are in [GGM]. Recently, designing pseudorandom functions from pseudorandom permutations (block ciphers) has been discussed in [HWKS].

Distributed Cryptography: We deal with systems where the capability to apply a cryptographic function (an RSA private function, in our case) is distributed in a threshold scheme. We have a group of l servers who securely share a private key d via shares $s_1, \ldots, s_l$. The servers are connected to a common broadcast medium C, called the communication channel, with the property that messages sent on C reach every party connected to it. We assume that each server has a local source of randomness and that the system is synchronized (and w.l.o.g. that servers act synchronously). When a quorum of t shareholders are available they can cooperate to compute the function, yet less than a quorum cannot break the system. Formally, in our security proofs we assume the corrupting adversary is non-adaptive throughout the lifetime of the system and is restricted to corrupt at most $t-1$ shareholders. Such an adversary is called a $(t-1)$-*restricted adversary*.

In a **threshold RSA system**, the function computation consists of a quorum of t servers who obtain an input m, generate partial signatures for m, and output these results to a *not* necessarily trusted combiner (which may simply be a designated server). The combiner, using the quorum of t partial signatures and input m generates the signature of the message, $m^d \bmod n$.[3] After signing a message, security is maintained in the sense that a quorum of t servers is again needed to sign a new message. When the signing operation is guaranteed (with high probability) to be polynomial-time under arbitrary malicious behavior of a $(t-1)$-restricted adversary, the system is called **robust**. If the system is maintained against a mobile adversary it is called **proactive**.

3 Pseudorandom Intermixing

The idea behind pseudorandom intermixing is simple. It involves entangling together (1) a cryptographic computation, and (2) a computation on pseudorandom numbers, such that the combined result is the same as that of the cryp-

[3] We note that since we do not put trust in the combiner, (1) it must be the case that the partial signatures do not provide any information that could help an adversary sign a new message, and (2) the combiner certainly should not hold any private keys.

tographic computation alone. In this paper, the cryptographic computation will generally be the computation of certain partial results of a distributed computation that will be summed together, and we will intermix (add in) pseudorandom numbers that sum to zero. Thus the result of the cryptographic computation will be unaffected by the pseudorandom numbers, although the "partial results" will be, in effect, randomized.

Here we give an example. Assume there is a set of users indexed by a set Λ, and each user i has a share s_i of a secret s where $s = \sum_{i\in\Lambda} s_i$. Also assume that each pair of users (i, j) shares a value $p_{i,j} = p_{j,i}$, which for now we may assume was chosen randomly from a large domain. It is easy to see that

$$\sum_{i\in\Lambda} \sum_{j\in\Lambda, j\neq i} p_{i,j} \cdot \text{sign}(j-i) = 0,$$

since each $p_{i,j}$ cancels $p_{j,i}$ (because they are added together with opposite signs). Now each user i can compute a pseudorandom intermixing value $r_i = \sum_{j\in\Lambda, j\neq i} p_{i,j} \cdot \text{sign}(j-i)$, and it follows that $\sum_{i\in\Lambda} r_i = 0$. User i may now compute an entangled output $s_i + r_i$, and we note that the sum of the entangled outputs is the secret:

$$\sum_{i\in\Lambda}(s_i + r_i) = \sum_{i\in\Lambda} s_i + \sum_{i\in\Lambda} r_i = s + 0 = s.$$

The final result is the same as the non-entangled computation, so why complicate the process? The issue is that during the process, the individual entangled outputs "look" random, and as we will show in the next section, this enables easier proofs of security in some involved distributed protocols.

We note that the $p_{i,j}$ values will actually be generated by a pseudorandom function. That is, each pair of users (i, j) will share a key $\sigma_{i,j} = \sigma_{j,i}$ to a pseudorandom function PRF, and they generate $p_{i,j} = \text{PRF}_{\sigma_{i,j}}(m)$ where m is a tag for the intermixing, such as a message to be signed. This tag serves a dual purpose of verifying that all users are working on the same computation at the same time.

The shared keys to PRF are relatively easy to generate using Diffie-Hellman key exchange. They can also be considered "disposable," since revealing them does not compromise the security of the main cryptographic function. We will see an example of this when we discuss the robust threshold RSA protocol.

Efficiency of size of keying information: It should be noted that even though extra keys are needed for pseudorandom intermixing with distributed RSA, our system is still storage efficient for practical implementations. In particular, we will deal with distributed cryptosystems where each server has a constant number of RSA key shares and thus a total of $O(l)$ RSA key shares must be stored. There are $O(l^2)$ pseudorandom intermixing keys shared in the entire system, but these are typically symmetric cryptography keys which are perhaps an order of magnitude smaller than the RSA key shares. (Recall that in [R] the threshold scheme requires $O(l^3)$ RSA keys in order to operate in a fault-tolerant threshold scheme without degradation of security or availability.)

We next investigate the application of pseudorandom intermixing to threshold RSA protocols. We remark that the method also applies to Discrete Log based systems in an analogous way. In Section 4.1 we convert the heuristic threshold RSA protocol from [DF91] into a provably secure protocol. In Section 4.2 we improve the efficiency of the secure threshold RSA protocol from [FGMY2]. In Section 5.1 we improve the efficiency of the robust threshold RSA protocol from [FGMY2]. A non-interactive proactive protocol in the fault free case is given in Section 5.2.

4 Applications to Threshold Cryptography

4.1 Pseudorandom Intermixing as Design Tool: Assuring Provable Security

In Crypto '91, [DF91] developed a heuristically secure threshold RSA scheme (see Appendix) in which servers do not need to communicate with each other to sign a message (i.e., it is a non-interactive threshold signing where outputs of servers which only have to know the working set, go to the combiner). The system is not known to be secure because it has never been proven whether the partial results sent from signing servers to the combiner S do not constitute sufficient information to allow for an adversary (which may hold S and up to $t-1$ servers) to sign new messages. Here we demonstrate how pseudorandom intermixing can take a protocol which is not known to be secure and make it into a secure one.

The protocol in [DF91] works in the following way. Server i obtains share s_i from a trusted key distributer. The scheme is heuristically secure since it is not known how an adversary with possession of $t-1$ or less servers can learn useful information that will compromise the system. Now, to sign a message m with group $\Lambda = \{i_1, \ldots, i_t\}$, each server i first generates $s''_{i,\Lambda}$ from s_i and the current active group Λ, (it holds that $d - 1 = \sum_{j\in\Lambda} s''_{j,\Lambda}$). Server i outputs $S_{m,i,\Lambda} = m^{s''_{i,\Lambda}} \bmod n$ to the Combiner. The Combiner can now compute signature $m^d \equiv m \prod_{j\in\Lambda} S_{m,j,\Lambda} \bmod n$. This latest process is not known to be simulatable.

We now modify [DF91] using pseudorandom intermixing .

1. Each pair of servers (i, j) share a private key $\sigma_{i,j} = \sigma_{j,i}$ for a pseudorandom function.
2. When applying a function on input m:
   The partial result now becomes $S_{m,i,\Lambda} = m^{s'_{m,i,\Lambda}} \bmod n$ where $s'_{m,j,\Lambda} = s''_{j,\Lambda} + [\ \sum_{v\in\Lambda\setminus\{j\}} \mathrm{sign}(j-v) \cdot \mathrm{PRF}_{\sigma_{j,v}}(m)\ ]$.
3. The combiner computes $m \prod_{j\in\Lambda} m^{s'_{m,j,\Lambda}} \equiv m^d \bmod n$ (since $\sum_{j\in\Lambda} s'_{m,j,\Lambda} = d-1$).

Fig. 1. Provably Secure DF91+pseudorandom intermixing

This slightly and modularly modified protocol (where fields in communicated messages did not change) is now secure:

Theorem 1. *Breaking the protocol of this subsection by a $(t-1)$-restricted adversary implies breaking RSA or that PRF is not a pseudorandom function family (i.e., the protocol is a secure threshold RSA protocol).*

Proof. (Sketch) We prove that if an adversary having possession of up to $t-1$ of the servers and the combiner can break the protocol then it can break RSA. We use a standard argument that the system can be simulated with input given to an attacker on the direct (non-distributed RSA scheme) who is allowed to probe the system on messages $m_1, \ldots, m_s$ and gets the signatures. The simulator then gets $\langle n, e, (m_1, m_1^d), \ldots, (m_s, m_s^d)\rangle$ where $s = \text{poly}(h)$ for security parameter $h = \log(n)$. Let $\Lambda_i = \{j_{i,1}, \ldots, j_{i,t}\}$ be the set of servers working together to sign message m_i. Moreover, let the adversary control the combining functions as well as servers indexed by A with shares of d such that $|A| \leq t-1$. We note that the distribution of the adversary receiving up to $t-1$ shares is simulatable.

When $|\Lambda_i \cap A| = t-1$ then trivially the simulator can compute $m_i^d/(m_i * \prod_{i\in A} m_i^{s'_{m_i,i,\Lambda_i}}) \bmod n$, simulating the output of the server the adversary does not control. In fact, also [DF91] can be simulated in this case. Now we discuss the case in which it is not known how to simulate in [DF91]. When $|\Lambda_i \cap A| \neq t-1$ then the output of any $i \notin A$ is pseudorandom due to the definition of S_{m_i,i,Λ_i} (except that the sum of all t of them adds up to the given $m^d/m \bmod n$). Hence, we can simulate by choosing random elements $R_1, \ldots, R_{t-1-|A|} \in_R Z_n^*$ as the outputs for $t-1-|A|$ of the servers that are not controlled by the adversary, and choosing the last server's output as $m_i^d/((m_i) \cdot (\prod_{i\in A} m_i^{s'_{m_i,i,\Lambda_i}}) \cdot (\prod_{j=1..t-1-|A|} R_j)) \bmod n$.

Now if the adversary, given the simulation, can break the system, then either: the simulation is indistinguishable from a real attack (the functions are pseudorandom), and breaking the distributed system implies breaking RSA. Otherwise, if the probability of breaking during the simulation is different from the probability of breaking in the actual attack differs by more than a negligible amount, this can be turned into a distinguisher for the "assumed pseudorandom function" which contradicts its pseudorandomness.

We note that if corruption is monotonic (rather than static where all corruption are at the start) we have to restart the simulation after each corruption. The above assumed the simulation stage after all corruptions are known.

Another nice property of the above modification is that there is no change in the interaction or messages used in the protocol. This is very important in cases where an old heuristic protocol has been implemented and the new secure one can be easily retrofitted .

4.2 Intermixing Assures Efficiency of Secure Threshold RSA

Next we describe the secure threshold protocol in [FGMY2]. This protocol has a setup phase (as in Figure 2, but without Step 3) where each server j obtains share s_j from a trusted dealer or via a distributed key generation protocol.

Now, to sign a message m in the secure threshold RSA protocol with group $\Lambda = \{i_1, \ldots, i_t\}$, each server j first generates $s_j z_{j,\Lambda}$ (where $z_{j,\Lambda}$ is a constant used for polynomial interpolation and is computable given j and Λ). Now it holds that $d = P + \sum_{j\in\Lambda}(s_j z_{j,\Lambda} + R_{j,m,\Lambda})$ where $R_{j,m,\Lambda}$ is a random string jointly generated (as discussed below) by the servers in Λ for server j. Server j outputs $S_{m,j,\Lambda} = m^{s_j z_{j,\Lambda} + R_{j,m,\Lambda}}$. The Combiner can now compute signature $m^d \equiv m^P \prod_{j\in\Lambda} S_{m,j,\Lambda} \bmod n$.

The generation of the $R_{j,m,\Lambda}$ values is a communication intensive protocol. Each party $j \in \Lambda$ first commits to $|\Lambda|$ random values via an information theoretically secure commitment scheme. This requires at least $2|\Lambda|$ exponentiations. Even more, a zero-knowledge proof is used to demonstrate that each commitment was performed correctly. Finally, there is a private transmission of the random values from each shareholder to the other shareholders in Λ. The idea is to change the polynomial sharing into a additive t-out-of-t sharing inside the group (changing the secret representation method is crucial in the work). Fortunately, the $R_{j,m,\Lambda}$ values can be cached, and thus they only need to be computed once for each Λ. However, this may require the shareholder to store significant amount of sensitive data.

Setup: The following one time setup is performed.

1. The (centralized/distributed) dealer generates an RSA with a public key (e, n) and a private key d.
2. Using the extended Euclidean algorithm, the dealer computes P, s' such $1 = eP + \frac{L^2}{H^2}s'$ where $L = l!$ and $H = \gcd(e, L)$. Note that $d \equiv P + L^2 \cdot k' \bmod \phi(n)$ where $k' \equiv ds'H^{-2} \bmod \phi(n)$. The dealer chooses a random polynomial $A(x) = A_0 + A_1 x + \cdots + A_{t-1}x^{t-1}$, such that $A(0) = A_0 = L^2 \cdot k'$ and $A_j \in_R \{0, L, \ldots, 2L^3 n^{2+\epsilon} t\}$ for $1 \le j \le t-1$. (All operations are performed over the integers.) All servers receive public point P, and each server i with public interpolation point $x_i = i$ receives secretly shadow $s_i = A(x_i) \in Z$.
3. **(Pseudorandom intermixing setup)** Each pair of servers (i, j) obtains a shared secret intermixing key $\sigma_{i,j} = \sigma_{j,i}$ for the pseudorandom generator. (In a distributed key generation protocol this can be performed via Diffie-Hellman key exchange [DH] with added authentication or some other shared randomness generation technique, but for now one may simply assume the keys are generated by the dealer.)

Notation: $z_{j,\Lambda} = \prod_{v\in\Lambda\setminus\{j\}}(x_j - x_v)^{-1}(0 - x_v)$

Fig. 2. Setup for a secure threshold RSA protocol with pseudorandom intermixing

Using pseudorandom intermixing, we may replace this interactive process of "generating randomness" with a non-interactive process. Moreover, the new process will require significantly less local computation.

For the new process we need a new setup protocol, shown in Figure 2. Observe that step 3 is the setup for the pseudorandom intermixing; the rest is the same as in [FGMY2].[4] The new signing protocol is shown in Figure 3. Observe that step 1 uses pseudorandom intermixing to create the "random-looking values" that were previously created through an interactive protocol.

Signing (operational) Phase: Let Λ where $|\Lambda| = t$ be the servers designated to participate in the signing The Combiner sends a description of the set Λ to all members of Λ.

1. Each server j computes $s'_{m,j,\Lambda} = s_j \cdot z_{j,\Lambda} + \{ \sum_{v \in \Lambda \setminus \{j\}} \text{sign}(j - v) \cdot \text{PRF}_{\sigma_{j,v}}(m) \}$
2. Each server j transmits partial signature $S_{m,j,\Lambda} \equiv m^{s_{m,j,\Lambda}} \bmod n$ and transmits the result to the Combiner.
3. The Combiner computes the signature for m from the partial signatures, $S_m \equiv m^P \cdot \prod_{v \in \Lambda} S_{m,v,\Lambda} \bmod n$.
4. **(Validate implicit hand-shake)** The Combiner verifies: $(S_m)^e \stackrel{?}{\equiv} m \bmod n$.

Fig. 3. Secure non-interactive threshold RSA signature generation with pseudorandom intermixing

Theorem 2. *The non-interactive randomized signing protocol in this subsection produces a correct RSA signature corresponding to public key* (e, n).

Proof. Note for any j where $1 \le j \le l$, L is divisible by $l \cdot \prod_{i \in \{1,\ldots,j-1,j+1,\ldots,l\}} (x_j - x_i)$. Let $H = \gcd(e, L)$ and observe that H^{-1} exists modulo $\phi(n)$ because e^{-1} exists. Let $1 = eP + \frac{L^2}{H^2} s'$. Note $d \equiv P + L^2 \cdot k' \bmod \phi(n)$ where $k' \equiv ds' H^{-2} \bmod \phi(n)$. Hence, over the rationals $d - P = L^2 k' = \sum_{i \in \Lambda} s_i \cdot z_{i,\Lambda}$ by the property of Lagrange interpolation [Sh]. Moreover, it is computable over the integers since $s_i \cdot z_{i,\Lambda}$ is an integer because L divides s_i. Hence,

$$\begin{aligned} d - P &= \sum_{i \in \Lambda} s_i \cdot z_{i,\Lambda} \\ &= \sum_{i \in \Lambda} s_i \cdot z_{i,\Lambda} + \sum_{i \in \Lambda} \Big(\sum_{v \in \Lambda \setminus \{i\}} \text{sign}(i - v) \cdot \text{PRF}_{\sigma_{i,v}}(m) \Big) \end{aligned}$$

[4] Throughout our discussions we assume a trusted dealer in order to simplify our discussion. However, we should note that using [BF97,FMY] one can employ a distributed dealer procedure among the shareholders to initiate the current protocol, hence not relying on any single entity (dealer).

$$= \sum_{i \in \Lambda} \left(s_i \cdot z_{i,\Lambda} + \sum_{v \in \Lambda \setminus \{i\}} \text{sign}(i-v) \cdot \text{PRF}_{\sigma_{i,v}}(m) \right)$$
$$= \sum_{i \in \Lambda} s'_{i,m,\Lambda},$$

since $(\text{sign}(i-v) \cdot \text{PRF}_{\sigma_{i,v}}(m)) + (\text{sign}(v-i) \cdot \text{PRF}_{\sigma_{v,i}}(m)) = 0$. Therefore, $m^d \equiv m^{P+L^2k'} \equiv m^P \cdot \prod_{i \in \Lambda} m^{s'_{i,m,\Lambda}} \bmod n$.

We now discuss the security of the protocol described in Figures 2 and 3 in the stationary adversary model, in which an adversary corrupts servers statically (we may also allow corruption in a non-adaptive monotonic fashion and restart the simulation from scratch after each corruption (monotonic means that once the adversary corrupts a server the server remains corrupted throughout). Moreover, we assume the adversary is $(t-1)$-restricted. The proof is similar to that of the previous section.

Theorem 3. *The protocol of this subsection is a secure threshold RSA protocol, i.e., if a $(t-1)$-restricted adversary can break the protocol, then RSA can be broken or PRF is not a pseudorandom function family.*

Proof. (Sketch) First we prove security assuming each $\text{PRF}_{\sigma_{i,j}}$ is a truly random function. We use a standard argument that the system can be simulated with input $\langle n, e, (m_1, m_1^d), \ldots, (m_s, m_s^d) \rangle$ where $s = \text{poly}(h)$ with security parameter $h = \log(n)$. Let $\Lambda_i = \{j_{i,1}, \ldots, j_{i,t}\}$ be the set of servers working together to sign message m_i. Say the adversary controls the Combiner as well as servers indexed by A where $|A| \leq t-1$. The simulatability of the up to $t-1$ shares the adversary sees was shown in [FGMY2]. Moreover, it is trivial to simulate the distribution of the (random) keys $\{\sigma_{i,j}\}$. When $|\Lambda_i \cap A| = t-1$ then the simulator can trivially compute the output $m_i^d / ((m_i^P) \cdot (\prod_{j \in A} m_i^{s'_{m_i,j,\Lambda_i}})) \bmod n$ of the server the adversary doesn't control. When $|\Lambda_i \cap A| < t-1$ then the output of Server j ($j \notin A$) is random by the definition of S_{m_i,j,Λ_i} (except that the sum of all t of them adds up to $m^{d-P} \bmod n$). Hence, we can simulate by choosing random elements $R'_1, \ldots, R'_{t-1-|A|} \in_R Z_n^*$ as the outputs for $t-1-|A|$ of the servers that are not controlled by the adversary, and $m_i^d / ((m_i^P) \cdot (\prod_{i \in A} m_i^{s'_{m_i,i,\Lambda_i}}) \cdot (\prod_{j=1..t-1-|A|} R'_j)) \bmod n$ as the output of the last server not controlled by the adversary. Now if one can break this system then they can use the simulation to break RSA using well established proof techniques.

Thus we may assume that the probability of breaking the system, assuming each $\text{PRF}_{\sigma_{i,j}}$ is a truly random function, is negligible. Then it is easy to conclude that if one can break the system, then PRF is not a pseudorandom function family, since the simulation can be made into a polynomial time distinguisher between a function from PRF and a truly random function.

Additional Properties It is often desirable to have additional security properties incorporated into the design of cryptographic protocols. We briefly list

here some additional properties that pseudorandom intermixing provides in the protocol above.

Handshake The intermixing of the results from the pseudorandom functions into the signature computations generates a "t-wise hand-shake." This creates a self-awareness property where the absence of a server is detected as long as a single original server is present– it is particularly useful when the servers want to be sure who is participating in the computation. (The protocol in [FGMY2] does provide the handshake but only when the interactive protocol is performed on a per message basis.) Thus the handshake effectively provides "access control" for a distributed computation, where the correctness of the final result implies the correct "working set" performed the computation.

Isolation The pseudorandom intermixing used in generating the partial results provides some protection against timing, power and other side channel attacks [K,KJJ]. Recently in [DKLMQW], an actual implementation of the timing attack was performed and publicly reported. Due to the pseudorandomness in our system, an adversary which looks at the timing (or other side information) of the partial result computation and tries to use the timing channel to deduce the permanent share will fail, since exponentiation is performed using a pseudorandom exponent which differs for each message. (This essentially creates a "random key schedule" for each message.)

5 Application to Fault Tolerant Protocols

Next we investigate applications to robust and proactive protocols.

5.1 Efficiency of Robust Threshold RSA

In the secure threshold RSA protocol above, if a server misbehaves, it may be difficult for the combiner to ascertain *which* server has misbehaved. Trying all subsets of servers until a correct signature is computed may be very expensive (e.g., iteratively attempting to find a subset of $t = l/2 + 1$ honest servers out of l would be exponential in l). Thus we need to make the protocol "robust," meaning that the protocol should allow the correct signature to be computed efficiently even when up to $t-1$ parties misbehave. This generally involves adding a method to verify the partial computations of the servers.

Robust RSA protocols were first introduced in [FGY] and then [GJKR96]. However, we will continue to work with the protocol from [FGMY2], but using pseudorandom intermixing to improve the efficiency.

As in [FGMY2], robustness can be added to the protocol from Section 4.2 by including verification information for the partial results in Step 2 in Figure 3. However, we use pseudorandom intermixing to decrease communication by about half compared to [FGMY2]. The protocol is shown in Figure 4. The idea here is to commit to the pseudorandom intermixing keys. When there is an active dispute

(but not when a server is simply unavailable) the commitment is opened to resolve the dispute. Note that it is the key for the pseudorandom function that is opened, and not an RSA key share. This is precisely what we meant in Section 3 by these shared keys being "disposable." Revealing them does not reveal any RSA shares, and thus does not reduce the security of the RSA signature function. Note also that we do not degrade the size of the quorum needed to sign due to unavailability of servers.

- (Recall System setup) The (centralized/distributed) dealer chooses generators g, g_1 from Z_n^* of maximal order, and where the discrete log of g_1 base g is unknown. Then for each i the dealer publishes $g^{s_i \cdot L^2} \bmod n$. Finally intermixing keys $\sigma_{i,j}$ and $\sigma'_{i,j}$ are generated for $1 \leq i, j \leq l$.
- (Server i setup) Each Server i publishes commitments $g^{\sigma_{i,j}} g_1^r, g^{\sigma'_{i,j}} g_1^{r'}$ to its intermixing keys.
- For a failed attempt in signing a message m by Λ:
  - Server i publishes $S'_{m,i,\Lambda} \equiv g^{L^2 s'_{m,i,\Lambda}} \bmod n$ and $U_{m,i,\Lambda} \equiv \prod_{j \in \Lambda \setminus \{i\}} (g_1)^{\mathrm{PRF}_{\sigma'_{i,j}}(m) \cdot \pi'(i,j)}$ where $\pi'(i,j) = -\mathrm{sign}(i-j)$.
  - For $i \neq j$, Server i or j publishes $R_{m,i,j} \equiv g^{\mathrm{PRF}_{\sigma_{i,j}}(m) \cdot L^2} (g_1)^{\mathrm{PRF}_{\sigma'_{i,j}}(m)}$. (In each pair, only one of Server i or Server j needs to publish, and the other needs to verify.)
  - Dispute resolution: if there is a dispute between Server i and Server j, then they open up their commitments to $\sigma_{i,j}$ and $\sigma'_{i,j}$, and a correct one is used to complete the protocol.
  - Each server j verifies that for all $i \in \Lambda \setminus \{j\}$: $(S_i)^{V_{i,1}} \stackrel{?}{\equiv} (S'_{m,i,\Lambda} (U_{m,i,\Lambda})^{-1} \prod_{v \in \Lambda} (R_{m,i,v})^{\pi'(i,v)})^{V_{i,2}}$ where $V_{i,1} = \prod_{v \in \Lambda \setminus \{i\}} (0 - x_v)$ and $V_{i,2} = \prod_{v \in \Lambda \setminus \{i\}} (x_i - x_v)$.
  - If the verification does not pass then Server i is removed and a new Λ is chosen to perform the signature.
  - Each Server i performs proofs of knowledge with all other servers of the discrete logs of $S'_{m,i,\Lambda}$ base g^{L^2} and $U_{m,i,\Lambda}$ base g_1 (This may be a non-interactive version of a Fiat-Shamir/Schnorr style proof [FS,Sch], with security based on the "ideal hash" assumption [BR93]).
  - Each Server i proves his partial signature is correct using the robustness algorithm of [FGY] or [GJKR96] (for safe primes), with witness $S'_{m,i,\Lambda}$.
  - If a server is unable to perform any of the ZK proofs, or if it has simply stopped, it is removed and a new Λ is chosen to perform the signature.

Fig. 4. Verification of partial signatures in a secure threshold RSA protocol with pseudorandom intermixing

The protocol in Figure 4 is performed for locating misbehaving servers if signing with the sum-shares produced an invalid signature.

Theorem 4. *The protocol as described in this subsection is a robust threshold RSA protocol, i.e., if a* $(t-1)$*-restricted adversary can break the protocol or prevent a valid message from being signed, then RSA can be broken or PRF is not a pseudorandom function family.*

Proof. (Sketch)

Security We start by assuming that each $\mathrm{PRF}_{\sigma_{i,j}}$ is a truly random function. As in the proof of Theorem 3, we use a standard argument that the system can be simulated with input $\langle n, e, (m_1, m_1^d), \ldots, (m_s, m_s^d)\rangle$ where $s = \mathrm{poly}(h)$ with security parameter $h = \log(n)$. (We also assume all corruption are known due to rewinding of the simulation to start). Let $\Lambda_i = \{j_{i,1}, \ldots, j_{i,t}\}$ be the set of servers working together to sign message m_i. Say the adversary controls the Combiner as well as servers indexed by A where $|A| \leq t-1$. The simulation of shares, intermixing keys, and partial signatures is done just as in the proof of Theorem 3. The simulation of the S_j values is done as in [FGMY2], and actually it is shown that the simulator knows $S_j^{1/L}$.

When $|\Lambda_i \cap A| = t-1$, the simulator can trivially simulate the S'_{m,v,Λ_i} value of the server v the adversary doesn't control by computing $g^d/((g^P) \cdot (\prod_{j\in A} g^{L^2 s'_{m_i,j,\Lambda_i}}))$ mod n. When $|\Lambda_i \cap A| < t-1$ then the S'_{m,j,Λ_i} values for $j \notin A$ are random (except that the product of all t of them is g^{d-P} mod n). Hence, we can simulate by choosing random elements $R'_1, \ldots, R'_{t-1-|A|} \in_R Z_n^*$ as the S'_{m,j,Λ_i} values for $t-1-|A|$ of the servers that are not controlled by the adversary, and $g^d/((g^P) \cdot (\prod_{j\in A} g^{s'_{m_i,j,\Lambda_i}}) \cdot (\prod_{j=1..t-1-|A|} R'_j))$ mod n as the S'_{m,v,Λ_i} value of the last server not controlled by the adversary.

To simulate the $R_{m,j,v}$ and U_{m,j,Λ_i} values, we simply choose the $R_{m,j,v}$ values randomly, and compute the U_{m,j,Λ_i} values that fit the verification equations. We can do this using

$$U_{m,j,\Lambda_i} \equiv \left((S_j^{1/L})^{LV_{i,1}/V_{i,2}}\right)^{-1} S'_{m,j,\Lambda_i} \prod_{v\in\Lambda_i\setminus\{j\}} (R_{m,j,v})^{\pi'(j,v)} \bmod n,$$

since $V_{i,2}$ divides L. We also simulate the zero-knowledge proofs using their respective simulators.

Now if one can break this system then they can use the simulation to break RSA using well established proof techniques.

Thus we may assume that the probability of breaking the system, assuming each $\mathrm{PRF}_{\sigma_{i,j}}$ is a truly random function, is negligible. Then it is easy to conclude that if one can break the system, then PRF is not a pseudorandom function family, since the simulation can be made into a polynomial time distinguisher between the a function from PRF and a truly random function.

Robustness We start by assuming that each $\mathrm{PRF}_{\sigma_{i,j}}$ is a truly random function. Then we have to show that if an adversary causes the signing procedure to fail, we can break RSA. Given an RSA instance, we run the simulator from the

security proof above, except that we use the extractors for the ZK proofs of knowledge.

For signing of a message m to completely fail, it must be that the adversary is able to make sure the signature protocol on m fails without any server being detected as corrupted. (If a server is detected as corrupted, it will be replaced and the signature protocol will continue on a different subset of servers.) By the last ZK proof in the protocol, the exponent on the partial signature $m^{s'_{m,j,\Lambda}}$ of server j, matches the exponent of $S'_{m,j,\Lambda}$ base g^{L^2}. Also, for the signature to be incorrect, it must be that $\sum_{j\in\Lambda} s'_{m,j,\Lambda} \neq d-P$. Recall that we have extracted the exponents on $S'_{m,j,\Lambda}$ and $U_{m,j,\Lambda}$ for all $j \in \Lambda \cap A$ from the proofs of knowledge. (For convenience, call them s'_j and u_j.) We also know the exponents on each S_j for $j \in \Lambda \cap A$, (since those are the simulated shares), and we know the values of $r_{i,j} = \mathrm{PRF}_{\sigma_{i,j}}(m)$ and $r'_{i,j} = \mathrm{PRF}_{\sigma'_{i,j}}(m)$ for $i \in \Lambda \setminus A$ and $j \in \Lambda \cap A$, since server i for $i \in \Lambda \setminus A$ knows $\sigma_{i,j}$ and $\sigma'_{i,j}$. Then we have

$$\begin{aligned}
\prod_{i\in\Lambda\cap A} S_i^{V_{i,1}} &\equiv \prod_{i\in\Lambda\cap A} (g^{L^2 s_i})^{V_{i,1}} \\
&\equiv \prod_{i\in\Lambda\cap A} \left(S'_{m,i,\Lambda} U_{m,i,\Lambda}^{-1} \prod_{v\in\Lambda\setminus\{i\}} (R_{m,i,v})^{\pi'_{i,v}} \right)^{V_{i,2}} \\
&\equiv \prod_{i\in\Lambda\cap A} \left(g^{L^2 s'_i} (g_1^{u_i})^{-1} \prod_{v\in\Lambda\setminus A} (g^{r_{i,v}} g_1^{r'_{i,v}})^{\pi'_{i,v}} \prod_{v\in\Lambda\cap A\setminus\{i\}} (R_{i,v})^{\pi'_{i,v}} \right)^{V_{i,2}} \\
&\equiv \prod_{i\in\Lambda\cap A} \left(g^{L^2 s'_i} (g_1^{u_i})^{-1} \prod_{v\in\Lambda\setminus A} (g^{r_{i,v}} g_1^{r'_{i,v}})^{\pi'_{i,v}} \right)^{V_{i,2}} \mod n,
\end{aligned}$$

and thus

$$\begin{aligned}
&g^{L^2 \sum_{i\in\Lambda\cap A}(V_{i,1}/V_{i,2}) s_i - L^2 \sum_{i\in\Lambda\cap A} s'_i - \sum_{i\in\Lambda\cap A} \sum_{v\in\Lambda\setminus A} r_{i,v}\pi'_{i,v}} \\
&\equiv g_1^{-\sum_{i\in\Lambda\cap A} u_i + \sum_{i\in\Lambda\cap A} \sum_{v\in\Lambda\setminus A} r'_{i,v}\pi'_{i,v}} \mod n.
\end{aligned}$$

However, if $L^2 \sum_{i\in\Lambda\cap A}(V_{i,1}V_{i,2}) s_i - L^2 \sum_{i\in\Lambda\cap A} s'_i - \sum_{i\in\Lambda\cap A} \sum_{v\in\Lambda\setminus A} r_{i,v}\pi'_{i,v}$ were zero, then the signature would be correct, since the product of the partial signatures of servers $j \in \Lambda \setminus A$ is equal to

$$U \equiv \frac{m^{d-P}}{\prod_{i\in\Lambda\cap A} m^{L^2 s_i - \sum_{j\in\Lambda\setminus\{i\}} \mathrm{PRF}_{\sigma_{i,j}}(m)\mathrm{sign}(i-v)}}$$

$$\equiv \frac{m^{d-P}}{\prod_{i\in\Lambda\cap A} m^{L^2 s_i - \sum_{j\in\Lambda\setminus A} r_{i,j}(m)\mathrm{sign}(i-v)}} \mod n$$

and thus the product of all partial signatures would equal

$$\begin{aligned} & U \cdot \prod_{i \in \Lambda \cap A} m^{L^2 s'_i} \\ & \equiv U \cdot \prod_{i \in \Lambda \cap A} m^{L^2 (V_1(i)/V_2(i)) s_i + \sum_{j \in \Lambda \setminus A} r_{i,j}(m) \mathrm{sign}(i-v)} \\ & \equiv m^{d-P} \bmod n. \end{aligned}$$

Thus, we can compute exponents a, b such that $g^a \equiv g_1^b \bmod n$, which implies that we can factor (and thus break RSA).

Similar to the proof of security, we can conclude that if the probability of breaking RSA is negligible and one can break the system, then PRF is not a pseudorandom function family, since the simulation can be made into a polynomial time distinguisher between the a function from PRF and a truly random function.

Implementation consideration: Because of the added inefficiency burden we recommend that the system is run without robustness testing until misbehavior is detected (i.e., an invalid signature is produced). At that point, the robustness enhancements can be incorporated to determine misbehaving servers.

5.2 Proactiveness: Intermixing for Efficiency, No-interaction and Added-Control

There are a number of ways intermixing can help in proactive RSA systems.

1. We may engage in a full proactive update as in [FGMY2] but use pseudorandom intermixing to reduce communication. In addition to share resharing, the pseudorandom functions should be replaced by new ones during updates.
2. We can also employ some dynamic updates which are less costly. One such change is updating the pseudorandom functions only. This can be done interactively (authenticated Diffie Hellman key exchange) between the parties.
3. Proactivization can be used to dynamically add and remove parties (via an update). If we stick to adding and removing from the recently updated group of parties, we can do it also by deciding to employ/not-employ the intermixings shared with them. This is an access-control function which computes on keys (analogous to "control vectors" acting on DES keys [M]). It assures that limitations on the cooperation can be easily achieved with intermixings (using them as credentials).
4. Whereas full proactive refreshment of cryptographic tools is needed to assure that past corruptions (memories learned in the past) are forgotten (namely erased and become irrelevant), we can take "simpler" mechanisms to assure that future corruptions cannot learn the past. This is done by "forward refreshment" of the keys for intermixing. This will ease the simulation arguments as the "pseudorandom past" becomes random. This can be achieved by updating the pseudorandomness based on "current round information"

and in a non-interactive fashion. A tag (i.e., date, counter which can be agreed upon) and previous randomness is used to generate a new pseudo-randomness for intermixing followed by an update. This can sometimes be extended to a full proactive update and implies, for example:

Theorem 5. *Using the above technique of forward refreshment with [FGMY] we are able to achieve fault-free non-interactive (e.g., all parties are honest and active in a round) full proactivization.*

This is the first time that such non-interactive maintenance is possible. It can be derived from [FGMY] (the first proactive RSA solution based on families of committees) by sharing pairwise intermixing where in each committee in a family one adds and the other subtracts using their new locally refreshed pseudorandomness (which is derived from the old key applied to a global tag which can be the global public state of this round). The new intermixing keys generate new shares, which is then followed by a non-interactive verification. At first glance this looks impossible since the adversary moves around in the system. But, notice that an adversary moves out of a server after it is detected in the mobile adversary model (otherwise silent spreading without limits is possible). Thus, when it is detected it is accounted for as a fault which, in turn, causes an interactive refresh which disconnects the past (due to perfect forward secrecy etc.). This amortization of "interaction" against "faults" enables the proof of the Theorem.

Let us give a simple example based on [F89], in which the secret key $d = s_1 + \ldots + s_t$ (it is used as a procedure for a "family" in [FGMY]). It demonstrates how proactive update can be done where if there are no faults the update is non-interactive. The new shares now become $s'_i = s_i + \sum_{j=1..i-1,i+1,t} \text{sign}(i-j)\text{PRF}_{\sigma_{i,j}}(tag)$ (we factor in the available "honest" pseudorandomness applied to a current "tag"). Before changing the s'_i a signature for some tag can be tested with the new shares. For [FGMY] there are many such sets $s_1, \ldots, s_u$ (held by a family of servers) such that they sum up to d and, moreover, more than one server can possess s_i. Proactive update in this system is a "sum-to-sum" process which takes the l-out-of-l to a different such system (within a family). For robustness of the update, a commitment to $\sigma_{i,j}$ is published as before. Moreover, the distributor (a single dealer or a distributed one) had published g^{s_i}. Publication of commitment to $\text{PRF}_{\sigma_{i,j}}(tag)$ is provided by i (and j) using say $C_{i,j} \equiv g^{\text{PRF}_{\sigma_{i,j}}(tag)}$ (one broadcast, to help update – but no more interaction is needed). Entities i and j may now have a dispute in case they disagree and value is opened if necessary (one will be wrong and removed, we will need to update from a different family if an entire committee is found corrupt). Each i also publishes $g^{s'_i}$ (for the next round using his share) and the following verification is made by each v (within a family with dispute phase if necessary) $g^{s'_i} \equiv g^{s_i} \prod_{j \in \Lambda \setminus \{i\}} (g^{\text{PRF}_{\sigma_{i,j}(tag)}})^{\text{sign}(i-j)}$. If no dispute s'_i is now used.

6 Conclusions

A new technique that entangles algebraic computations with shared pseudorandomness and which applies to numerous existing protocols has been described. We have shown that it can be a design tool for incrementally achieving/ retrofitting correctness, efficiency, provability, and extended functionality in distributed cryptographic protocols. It also allows for better isolation (individualization) of computations and added (access)-control. Inclusion of the tool in various other distributed cryptographic protocols may result in further improvements.

References

[B97] D. Beaver, Commodity-based Cryptography, In the 29-th STOC, 446–455, 1997. 309

[BR93] M. Bellare, and R. Rogaway, *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, ACM Conference on Computer and Communications Security, 1993. 318

[BR94] M. Bellare, and R. Rogaway, *Optimal Asymmetric Encryption*, Eurocrypt 94. 92-111.

[Bl] R. Blakley, *Safeguarding Cryptographic Keys*, FIPS Con. Proc (v. 48), 1979, pp. 313–317. 307

[BM] G.R. Blakley and C. Meadows, *Security of Ramp Schemes*, Crypto 84, LNCS 196, 242–268. 308

[BF97] D. Boneh and M. Franklin, *Efficient Generation of Shared RSA Keys*, Crypto 97 proceedings. 315

[B88] C. Boyd, *Digital Multisignatures*, IMA Conference on Cryptography and Coding, Claredon Press, 241–246, (Eds. H. Baker and F. Piper), 1989. 307

[D92] Y. Desmedt. Threshold cryptosystems. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology—AUSCRYPT '92*, volume 718 of *Lecture Notes in Computer Science*, pages 3–14. 1992, Springer-Verlag. 307

[DKLMQW] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestreacute, J.-J. Quisquater and J.-L. Willems, *A Practical Implementation of the Timing Attack*, Cardis '98. 317

[DDFY] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, *How to Share a Function Securely*, ACM STOC '94, pp. 522-533. 307

[DF91] Y. Desmedt and Y. Frankel, *Shared Generation of Authenticators and Signatures* Advances in Cryptology-Crypto '91, pp. 457-469. Springer-Verlag. 307, 308, 312, 313

[DH] W. Diffie and M. Hellman, *New Directions in Cryptography* , IEEE Trans. on Information Theory 22 (6), 1976, pp. 644-654. 310, 314

[F] P. Feldman, *A Practical Scheme for Non-Interactive Verifiable Secret Sharing*, FOCS '87, pp.427-437. 308

[FS] A. Fiat and A. Shamir, *How to prove yourself: Practical solutions to identification and signature problems*, Crypto'86, pp. 186-194. 318

[F89] Y. Frankel, *A practical protocol for large group oriented networks*, In J. J. Quisquater and J. Vandewalle, editor, *Advances in Cryptology, Proc. of Eurocrypt '89, (Lecture Notes in Computer Science 773)*, Springer-Verlarg, pp. 56-61. 307, 322

[FY98] Y. Frankel and M. Yung. Distributed public-key cryptosystems. In H. Imai and Y. Zheng, editors, *Advances in Public Key Cryptography—PKC '98*, volume 1431 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, Feb. 1998. invited talk. 307

[FGMY] Y. Frankel, P. Gemmell, P. MacKenzie and M. Yung. *Proactive RSA*, crypto 97. 307, 322

[FGMY2] Y. Frankel, P. Gemmell, P. MacKenzie and M. Yung. *Optimal Resilient Proactive Public-Key Systems*, FOCS 97. 307, 308, 312, 314, 315, 316, 317, 319, 321

[FGY] Y. Frankel, P. Gemmell and M. Yung, *Witness Based Cryptographic Program Checking and Robust Function Sharing*. STOC96, pp. 499-508. 307, 317, 318

[FMY] Y. Frankel, P. MacKenzie and M. Yung. *Robust Distributed Efficient RSA-key Generation*, manuscript. 315

[GJKR96] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, *Robust Threshold RSA*, Crypto96, pp. 157-172. 307, 308, 317, 318

[GI99] N. Gilboa and Y. Ishai, *Compressing Cryptographic Resources*, Crypto99, pp. 591-608. 309

[GGM] O. Goldreich, S. Goldwasser and S. Micali, *How to construct random functions*, J. Comm. Sci. 28 (1984), pp. 270-299. 310

[HWKS] C. Hall, D. Wagner, J. Kelsey, and B. Schneier, *Building PRFs from PRPs*, Proceedings of Crypto '98, 1998, Springer-Verlag, pp370–389. 310

[HJJKY] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, M. Yung, *Proactive Public-Key and Signature Schemes* Proceedings of the Fourth Annual ACM Conference on Computer and Communications Security, CCS '97.

[K] P. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSA and Other Systems*, Crypto96. 317

[KJJ] P. Kocher, J. Jaffe and B. Jun, *Differential Power Analysis*, Crypto99. 317

[M] S. M. Matyas, *Key processing with control vectors*, Journal of Cryptology, 3 (2), pp. 113-136, 1991. 321

[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. 307

[R] T. Rabin, *A simplified approach to Threshold and Proactive RSA*, Proceedings of Crypto 98, Springer-Verlag, 1998, pp. 89–104. 307, 308, 311

[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. 307

[Sch] C. P. Schnorr, *Efficient identification and signatures for smart cards*, Crypto'89, pp. 239-252. 318

[Sh] A. Shamir, *How to share a secret*, Comm. of ACM, 22 (1979), pp. 612-613. 307, 315

[Sho] V. Shoup, Personal communication. 307

A Brief Description of Desmedt-Frankel

The following one-time setup is performed.

– The dealer generates an RSA public key (e, n) and the associated private key d. The modulus n is generated such that it is a composite of two safe primes, $p = 2p' + 1$ and $q = 2q' + 1$ where p', q' are primes. Let $R = 2p'q'$.

- The dealer chooses a random polynomial $f(x) = A_0 + A_1 x + \cdots + A_{t-1} x^{t-1}$, such that $A(0) = d - 1$ and $A_i \in_R Z_R$.
- Each Server i is assigned interpolation point $x_i = 2i$ and receives share

$$s_i \equiv f(x_i) \cdot \left(\prod_{j \in \{1,\ldots,l\} \setminus \{i\}} (x_j - x_i) \right)^{-1} \bmod R$$

over a private channel.

A set of servers Λ can sign a message m by participating in the signing protocol in Figure 3, with $P = 1$, $\mathrm{PRF}_{\sigma_{i,j}}(m) = 0$ for all $i, j \in \Lambda$, and $z_{i,\Lambda} = (\prod_{j \in \{1,\ldots,l\} \setminus \Lambda} (x_i - x_j))(\prod_{j \in \Lambda \setminus \{i\}} (0 - x_j))$.