Threshold Cryptography Secure Against the Adaptive Adversary, Concurrently

Anna Lysyanskaya[*]
MIT LCS, NE43-334
545 Technology Sq., Cambridge, MA 02139 USA
anna@theory.lcs.mit.edu

May 12, 2000

Abstract

A threshold cryptosystem or signature scheme is a system with n participants where an honest majority can successfully decrypt a message or issue a signature, but where the security and functionality properties of the system are retained even as the adversary corrupts up to t players. We present the novel technique of a committed proof, which is a new general tool that enables security of threshold cryptosystems in the presence of the adaptive adversary. We also put forward a new measure of security for threshold schemes secure in the adaptive adversary model: security under concurrent composition. Using committed proofs, we construct concurrently and adaptively secure threshold protocols for a variety of cryptographic applications. In particular, based on the recent scheme by Cramer-Shoup, we construct adaptively secure threshold cryptosystems secure against adaptive chosen ciphertext attack under the DDH intractability assumption.

Keywords: threshold cryptosystem, threshold signature scheme, adaptive security, proactive security, concurrency.

1 Introduction

The threshold setting [DF89] generalizes a cryptosystem or signature scheme in the sense that decryption or signing is performed by a group of servers

[*]This research was carried out while the author was visiting IBM Research Lab, Zürich, Switzerland

instead of just one. This setting is non-trivial because some minority of the servers may be malicious. Threshold cryptography yields implementation of one trusted party, under the assumption that the majority of some servers can be trusted. In turn, having a trusted party is useful for tasks such user certification [Bra99], fair exchange [AWS99], decryption of sensitive information, etc.

The model The threshold model consists of n partially synchronized servers (i.e. servers who have clocks that differ by a fixed amount), out of which t may become malicious over the course of the lifetime of the system. Without loss of generality, it is assumed that the t malicious servers will get corrupted by a single adversary. The players have access to an authenticated broadcast channel. In addition, there are communication links between any pair of players.

Adaptive vs. Static Adversary The focus of this paper is the security against the adaptive adversary. The adaptive adversary in a threshold system is an adversary that corrupts the players based on all the information available to it from the run of the protocol.

In contrast, the better-studied static adversary is an adversary that corrupts players independently on its view of the protocol. Most of the threshold systems in the literature are secure against static adversaries only [FGMY97, Rab98, SG98, Sho99b], or can only tolerate a very low threshold for adaptive adversaries. For example, the Canetti-Goldwasser threshold implementation [CG99] of the Cramer-Shoup cryptosystem [CS98], is secure and robust against up to $O(\sqrt{n})$ adaptively corrupted servers.

Security against the adaptive adversary is preferable to security against the static adversary because (1) the adaptive adversary is strictly more powerful than the static one [CFGN96, CDD$^+$99]; (2) the adaptive attack is a practical attack.

Previous work There are two works in the literature that present practical constructions of a threshold cryptographic system secure against adaptive adversary: the results of Canetti et al. [CGJ$^+$99a] with DSS and RSA, and that of Frankel et al. [FMY99b] also on RSA. However, the solutions presented in these works are not immediately applicable to other cryptosystems and signature schemes, and are not known to be secure under concurrent composition.

The committed proof technique We develop a new, general technique of a committed zero-knowledge proof of knowledge. Zero-knowledge proofs are useful in threshold protocols to guarantee the robustness property: a server is required to prove that his share of the decryption or signature is valid. The stumbling block of guaranteeing robustness in the presence of the adaptive adversary used to be the fact that if a server is corrupted while this proof is carried out, the adversary learns more information than the desired security allows. Our key observation is that by proving validity first, and revealing the decryption or signature share later, we allow a server to erase some temporary secret in the meantime, and prevent the adversary from learning more than the definition of security allows. We discuss this technique in section 2.

One may imagine that the servers involved in a threshold scheme might also be involved in a large number of other threshold schemes and multi-party computations. Also, within a single scheme, they might want to perform several tasks concurrently. For that reason, we want a threshold scheme to have the property that it is composable with other such schemes under concurrent executions. It turns out that our committed proof technique makes it possible to preserve security under concurrent composition.

We illustrate the generality of our techniques by adapting the (statically secure) Canetti-Goldwasser [CG99] implementation of threshold Cramer-Shoup cryptosystem [CS98] such that it becomes adaptively secure. If carried out on-line, it is secure for $t \leq n/2$, and in the off-line case, the security is achieved for $t \leq n/3$. In this implementation, the amount of communication between players, and the amount of computation they need to invest into the process, is on the same order as that of the Canetti-Goldwasser's [CG99] scheme. We also make our threshold cryptosystem proactive.

Organization of this paper Section 2 is dedicated to the committed proof technique. Section 3 is an overview of threshold protocols. Section 4 introduces several building blocks that will be used often. Section 5 gives our implementation of the threshold Cramer-Shoup cryptosystem.

2 The committed proof

In this section, we present the notion of a committed proof, which is a zero-knowledge proof of knowledge that is carried out *in a committed form.* The verifier does not learn the statement that is being proven until the very last round of the protocol. For definitions and discussions of zero-knowledge proof systems and zero-knowledge proof of knowledge systems, we refer the reader to a treatment by Goldreich [Gol98] and by Bellare and Goldreich [BG92].

Suppose we are given three-step public-coin honest-verifier zero-knowledge proof of knowledge system Z [BG92] for language L, as follows:

1. The proof system has perfect completeness and soundness $2^{-\Omega(k)}$.
2. The prover's input is $x \in L$, a witness w, and some randomness r.
3. The random coins R are tossed after the prover issues the first message.
4. Algorithms $P_1(x, w, r)$, and $P_2(x, w, r, R)$ generate the first and second messages of the prover.
5. The verifier runs algorithm $Ver(x, m_1, R, m_2)$ to determine whether to accept or reject.
6. The simulator algorithm SIM used for proving the zero-knowledge property of Z, has the property that for all inputs $R \in \{0,1\}^k$, it generates an accepting transcript (m_1, R, m_2) indistinguishable from a transcript of a conversation with the real prover.
7. The knowledge extractor algorithm KE for Z has the property that, for some constant c, on input $(x, m_1, R, R', \ldots, R^{(c)}, m_2, m_2', \ldots, m_2^{(c)})$ such that $R \neq R' \neq \ldots \neq R^{(c)}$ and Ver accepts all transcripts (x, m_1, R, m_2), (x, m_1, R', m_2'), $\ldots$, $(x, m_1, R^{(c)}, m_2^{(c)})$, KE outputs a witness w with probability $1 - neg(k)$.

Such proof systems exist for all languages in NP, by a witness-preserving reduction to Hamiltonian cycles [Gol95]. In particular, for proving knowledge or equality of discrete logarithms or representations, such proof systems have perfect simulations and are well-studied and efficient [Bra99, Cam98].

Suppose that x for which the prover is demonstrating membership in L is unknown to the verifier. However, the verifier knows the distribution $\mathcal{D}$ from which x has been sampled. Moreover, $\mathcal{D}$ has the property that there is an efficiently samplable joint distribution $(\mathcal{W}, \mathcal{D})$ from which pairs (w, x) are sampled, such that w is a witness for the statement $x \in L$. For example, x can be a tuple $(G_q,\ g,\ h,\ y)$ and statement $x \in L$ means that y is an element in G_q that can be represented in bases g and h. When we sample $\mathcal{D}$, we can first generate a random $\alpha, \beta \in \mathbb{Z}_q$, then and then set $w = (\alpha, \beta)$, and $y = g^\alpha h^\beta$.

Suppose we are given a *trapdoor* commitment scheme, i.e. a commitment scheme that has the property that for any instance of the commitment scheme, there exists a trapdoor σ the knowledge of which enables to open any commitment to an arbitrary value within some given domain.

For example, consider Pedersen commitment: an instance is a group G_q of order q in which the discrete logarithm problem is hard, with generators g and h and a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q^*$. The trapdoor $\sigma = \log_g h$. To commit to x, choose a random r and output $g^{\mathcal{H}(x)} h^r$. To open the commitment, reveal x and r. If σ is known, it is easy to see that a commitment can be opened to any x. Note that if we are not given a collision-resistant hash function, then the prover can still commit to his input x and the first message of the proof, but this commitment will have to use some special encoding of x and will be larger.

How can we create a simulator such that σ is known to it? In multi-party systems, we can have an instance of the commitment scheme generated as part of the set-up for the system; then it will follow from the properties of multi-party computation that a simulator will know σ. We discuss such a protocol in section 4.2. In two-party protocols, σ can be a value known to the verifier, but not the prover; the simulator with black-box access to the verifier will then have to *extract* σ from the verifier.

Using trapdoor commitments, the prover can execute the proof *without revealing x to the verifier until the very end of the proof.* Consider the protocol in figure 1 between a prover and a verifier. The protocol uses Pedersen commitment, but any trapdoor commitment can be used instead.

Common inputs: (G_q, g, h): an instance of Pedersen commitment.
Prover's inputs: statement $x \in \mathcal{D}$, witness w, random input r.
Verifier's goal: obtain x s.t. prover knows a witness to "$x \in L$."

$P \longrightarrow V$ Prover computes $m_1 = P_1(x, w, r)$, chooses random r_1 and sends $M_1 = g^{\mathcal{H}(x,m_1)} h^{r_1}$.

$P \longleftarrow V$ Verifier tosses random coins R and sends them to the prover.

$P \longrightarrow V$ Prover computes $m_2 = P_2(x, w, r, R)$, chooses random r_2 and sends $M_2 = g^{\mathcal{H}(m_2)} h^{r_2}$. Prover erases w.

$P \longrightarrow V$ Prover sends x, m_1, m_2, r_1, r_2, i.e. opens commitments M_1, M_2.

Acceptance: The verifier accepts if M_1 is a valid commitment to x and m_1, M_2 is a valid commitment to m_2, and $Ver(x, m_1, R, m_2)$ accepts.

Figure 1: Committed proof

Lemma 1 (Completeness): *This protocol has perfect completeness.*

Proof: We get completeness for free from proof system Z. $\square$

Lemma 2 (Zero-knowledge) *This protocol is zero-knowledge for any verifier.*

Proof: The lemma follows from the fact that for a simulator that knows $\log_g h$, the commitments M_1 and M_2 are not binding, and so the simulator can reveal x, message m_1 and response m_2 in the very end, when it already knows the challenge R, by property 6 of proof system Z. $\square$

Note: Notice that the original proof system Z was zero-knowledge for the public-coin model only, while the proof system we obtain is zero-knowledge for *any* verifier. (We achieve this because of a preprocessing step that generates h.)

Lemma 3 (Concurrent composition) *This protocol remains secure when executed concurrently (i.e. with an arbitrary interleaving of steps) with arbitrarily many invocations of itself or of any other concurrently composable protocols.*

Proof: The lemma follows from the fact that the above simulator that exhibits the zero-knowledge property does not need to rewind the verifier. $\square$

Lemma 4 (Soundness and knowledge extraction) *If the discrete logarithm problem is hard, and the hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q{}^*$ is collision-resistant, then for this protocol there exists a polynomial-time knowledge extractor such that if the verifier accepts with non-negligible probability, then with probability $1 - neg(k)$ the knowledge extractor learns the witness w for x that the prover possesses.*

Proof: We will exhibit a knowledge extractor which, with black-box access to the prover that induces the verifier to accept with non-negligible probability, either extracts a witness for x or computes the discrete logarithm of h to the base g, or finds a collision in $\mathcal{H}$. Clearly this is sufficient to prove the lemma.

The extractor runs the prover and obtains the x, as well as m_1, R, m_2 and M_1, r_1, M_2, r_2. Now the extractor rewinds the prover to step 2 of the protocol and issues a challenge $R' \neq R$. Running the protocol to the end allows the verifier to obtain x', as well as m_1', m_2', r_1', r_2' and M_2'. Note that since the prover replies with non-negligible probability, with enough rewindings, we will get as many replies from him as the knowledge extractor KE of proof system Z may need.

Suppose $x \neq x'$. Then either $x = \mathcal{H}(x, m_1) \neq \mathcal{H}(x', m_1') = x'$ or we have found a collision in the hash function. If the latter, we have the desired contradiction. Otherwise, $g^x h^{r_1} = M_1 = g^{x'} h^{r_1'}$, and so we can compute $\log_g h$.

Now suppose $x = x'$. Then, by the same argument as above, $m_1 = m_1'$ or we find a collision or compute discrete log. Then since m_2 is a valid response to challenge R and so is m_2' to challenge R', it follows from the fact that Z is a proof of knowledge that we can extract a witness for x by using KE. □

Finally, lemma 5 below is the key to why a committed proof is instrumental for designing protocols that are secure against the adaptive adversary. It captures the counter-intuitive fact that the prover can be attacked in the middle of the proof, but the adversary still learns nothing, i.e. the zero-knowledge property of the whole game is retained! The only condition required is that the distribution $(\mathcal{W}, \mathcal{D})$ that captures the adversary's *a priori* information about the distribution that x and witness w come from, be efficiently samplable.

Lemma 5 (Security against corruption) *If the prover is corrupted by the adversary in the middle of the proof, everything that the adversary learns can be accomplished either by revealing x, or by sampling $(\mathcal{W}, \mathcal{D})$.*

Proof: We prove the claim by exhibiting a simulator $\mathcal{S}$ which generates the adversary's view of the corruption. Suppose the adversary decides to corrupt the prover just before the end of step 2. Then $\mathcal{S}$ samples $(\mathcal{W}, \mathcal{D})$ and obtains a witness w' for an x'. $\mathcal{S}$ the generates a random r and, using trapdoor $\sigma = \log_g h$ computes $m_1' = P_1(x, w, r)$ and r_1' such that $M_1 = g^{\mathcal{H}(x', m_1')} h^{r_1'}$, as well as $m_2' = P_2(x, w, r, R)$ and r_2' such that $M_2 = g^{\mathcal{H}(m_2')} h^{r_2'}$. Reveal w', x', r, r_1', r_2' to the adversary. These values are distributed correctly since w' and x' come from distribution $(\mathcal{W}, \mathcal{D})$ and r, r_1', r_2' are all random values.

Suppose the adversary decides to corrupt the prover at some step before the end of step 2. Then it is clear that $\mathcal{S}$ will just have to reveal a subset of the values above (depending on whether M_1 and M_2 have been issued yet).

Suppose the adversary corrupts the prover after the end of step 2, i.e. after w was erased. Since w is erased, the adversary learns nothing more than what the verifier can learn. Thus, $\mathcal{S}$ just runs the simulator we have constructed for proving the zero-knowledge property. □

As we will see in section 5, this property of a committed proof allows us to create a perfect and never failing simulation of the adversary's view, which implies full concurrency of the erasure-enabled threshold cryptosystems we propose.

3 Threshold protocols: overview

Our goal is to build a cryptographic system which is secure, robust, and proactive and also co-exists with other such systems in the concurrent setting.

The servers and their communication model The servers are interactive probabilistic Turing machines that can erase their memory. This assumption is necessary because the only known way of achieving proactiveness of a threshold system is through the use of erasures. They are in the partially synchronous communication model. There is an authenticated broadcast channel from each player to all other players, and any pair of players is connected by a dedicated secure private channel. Note that since we allow erasures, the assumption on the security of the private channels is not very dramatic. For example, private channels can be implemented by refreshing the secret keys used by all pair of servers after each message is sent (i.e. append a new secret key to the end of each message, and erase the old secret key). The time it takes for a message to be delivered is known and we assume it is not long.

Protocol properties Let l be the security parameter. Let n be the number of players. Let the parameter t denote the maximum number of players that the adversary is able to control. Informally, we say that a threshold procedure is t-secure if, given the input and output of the procedure, we can simulate the view of any t (possibly malicious) players that participate in it. We say that a threshold procedure is t-robust if, for any set of players $MALICIOUS$ of size at most t, the distribution of the output of the procedure is independent on whether the set $MALICIOUS$ behaves according to the prescribed protocol, or deviates from it arbitrarily. We say that a threshold system is secure and robust if any valid sequence of procedures in the system is a secure and robust procedure. We say that a threshold system is proactive if it retains its security over time in the presence of an adversary who, over time, loses control over some previously corrupted players, but gains control over new players such that the number of players controlled by the adversary at any given time is at most t. We say that a threshold system is concurrently secure if it remains secure and robust when executed concurrently with other procedures.

4 Building blocks for our protocols

In this section, we exhibit building blocks that lead to the construction of discrete-logarithm-based threshold cryptographic applications.

4.1 Computational assumptions

We make the following computational assumptions: (1) We are given a group G_q of prime order q in which the decisional Diffie-Hellman problem [Bon98, Wol99] is hard. For example, it is believed that this can be achieved by choosing a prime p such that $p = 2q + 1$ for a prime q. Then take the subgroup of $\mathbb{Z}_p^*$ generated by an element g of order q. (2) We have a collision-resistant hash function $\mathcal{H}$ (or, at the expense of efficiency, we can make do with a universal one-way hash function family) [BR97, Sho99a].

4.2 Generating an auxiliary base

To enable the protocols that comprise our construction, the group G_q defined above must have two generators, g and h, such that the discrete logarithm of h to the base g in unknown to any proper subset of the n players. In order to guarantee this property, and also in order to be able to prove security of our protocols, we need to implement the generation of the auxiliary base h in a distributed fashion, as described below.

This is a special building block which will be executed once, in the setup phaze of the system, before any other protocol starts. Therefore the fact that the proof of security for this protocol requires a simulation with rewinding, does not imply that rewinding will be needed in any other protocols.

The generation of h must be done with the vital additional property that when this is a simulated run, instead of a real run (i.e. when we are showing that a simulator can be constructed that simulates the adversary's view, which is necessary to prove security), the simulator, which controls a majority of players, will get to know this value, or will get to know the representation of h in some bases $g_1, ..., g_l$ of its choice, or will even force h to be some value it wants, whichever the simulator needs for its purposes. This can be achieved using the techniques of unconditionally secure verifiable secret sharing, developed in Cramer et al. [CDD$^+$99]. (Note that even though their multi-party computation results are not applicable here because they are not efficient, a simple protocol such as VSS is efficient enough, and we have the same communication and adversary model.)

4.3 Generation of a verifiable shared random secret

In Figure 2, we include the well-known protocol Joint-RVSS [Ped91b, GJKR99] for joint verifiable sharing of a random secret, which is a basic building block of our protocols. We present it here for the purposes of complete presentation using notation that is useful for the presentation of the protocols that follow. Our notation is adopted from Jarecki and Lysyanskaya [JL00a].

Notation: We say that players generate $\mathsf{RVSS\text{-}data}_{t,g,h}[a]$ if they execute this protocol with generators g, h and polynomials of degree t. We index the data produced with labels a, α, using the associated Greek letter for polynomial shares.

Random sharing of 0 Note that by running the protocol above with value $t-1$ for the threshold, and adding the constraint that $f_a(0) = 0$, the players can obtain a polynomial sharing of a random polynomial of degree t.

4.4 Distributed coinflip

One use of Joint-RVSS is in a distributed coinflip protocol (Fig.3), whose security properties are formalized in Lemma 6. This lemma is useful also for other uses of Joint-RVSS, where unlike in the coinflip protocol, the generated secret is not explicitely reconstructed. The presentation in this section is also adopted from the presentation of Jarecki and Lysyanskaya [JL00a].

Lemma 6 *On input a random string r, the non-rewinding simulator* SIM *simulates the adversary's view of the distributed coinflip protocol that results in r.*

Proof: The simulator for the security proof is contained in figure 3. The simulator knows $\log_g h$, thus it need not decide on a_i's for players P_i it controls until it learns a_j for each player P_j that the adversary controls. (Note that the simulator can determine the adversary's value a_j by interpolating $f_{a_j}(i)$.) After that, the simulator assigns values a_i to the players in such a way that $\sum_{P_i \in Qual} a_i = a^*$. $\Box$

Protocol: (on inputs group G_q, generators g, h)
1. Each player P_i performs a Pedersen VSS of a random value a_i:
(a) P_i picks two t-deg. polynomials, $f_{a_i}(z) = \sum_{k=0}^{t} c_{ik} z^k$, $f_{\hat{a}_i}(z) = \sum_{k=0}^{t} \hat{c}_{ik} z^k$
Let $a_i = f_{a_i}(0)$ and $\hat{a}_i = f_{\hat{a}_i}(0)$ be the values shared by these polynomials
P_i broadcasts $C_{ik} = g^{c_{ik}} h^{\hat{c}_{ik}}$ for $k = 0..t$ which defines $F_{a_i}(z) = \prod_{k=0}^{t} (C_{ik})^{z^k}$
P_i sends to P_j shares $\alpha_{ij} = f_{a_i}(j)$, $\hat{\alpha}_{ij} = f_{\hat{a}_i}(j)$ for each $j = 1..n$
(b) Each P_j verifies if $g^{\alpha_{ij}} h^{\hat{\alpha}_{ji}} = F_{a_i}(j)$ for $i = 1..n$
If the check fails for any i, P_j broadcasts a *complaint* against P_i
(c) If P_j complained against P_i, P_i broadcasts α_{ij}, $\hat{\alpha}_{ij}$ and everyone verifies it.
If P_i fails this test or receives more than t complains, it is excluded from $Qual$
2. P_i sets his polynomial share of the generated secret a as
$\alpha_i = \sum_{P_j \in Qual} \alpha_{ji}$, and their associated randomness as $\hat{\alpha}_i = \sum_{P_j \in Qual} \hat{\alpha}_{ji}$

We label the data structure created by this protocol as $\mathsf{RVSS\text{-}data}_{t,g,h}[a]$:
Secret Information of each player P_i: (well-defined for $P_i \in Qual$)
– $a_i, \hat{a}_i$ his additive shares of the secret and its associated randomness
– $f_{a_i}, f_{\hat{a}_i}$ t-degree polynomials he used in sharing his additive share
– $\alpha_i, \hat{\alpha}_i$ his polynomial share of the secret and its associated randomness
– $\alpha_{ji}, \hat{\alpha}_{ji}$ his polynomial shares (and assoc. rand.) of $f_{a_j}, f_{\hat{a}_j}$ for $j = 1..n$
Public Information:
– the set $Qual \subseteq \{P_1, .., P_n\}$
– verification function $F_a : \mathbb{Z}_q \to \mathbb{Z}_p{}^*$ (see the implicit information below)
– verification functions $F_{a_i}(z) = g^{f_{a_i}(z)} h^{f_{\hat{a}_i}(z)}$ for $P_i \in Qual$
Secret Information Defined *Implicitly* (not stored by any player):
– secret sharing t-degree polynomials $f_a(z), f_{\hat{a}}(z)$ s.t. $\alpha_i = f_a(i)$, $\hat{\alpha}_i = f_{\hat{a}}(i)$,
$f_a(z) = \sum_{P_i \in Qual} f_{a_i}(z)$, $f_{\hat{a}}(z) = \sum_{P_i \in Qual} f_{\hat{a}_i}(z)$, and $F_a(z) = g^{f_a(z)} h^{f_{\hat{a}}(z)}$
– secret shared value $a = f_a(0)$ and its associated randomness $\hat{a} = f_{\hat{a}}(0)$

Figure 2: Joint-RVSS creates a secret sharing RVSS-data[a] of random secret $a \in \mathbb{Z}_q$

4.5 Simultaneous zero-knowledge proofs of knowledge

Our adaptive protocols, following the protocols of Canetti et al. [CGJ+99a], use *simultaneous* zero-knowledge proofs of knowledge to enable robustness efficiently. We describe this technique here, adapting the presentation due to Jarecki and Lysyanskaya [JL00a].

Consider any honest-verifier public-coin zero-knowledge proof of knowledge system (ZKPK) [BG92]. Say that the prover shows knowledge of witness w of a public relation $A = (y, x)$ for some value y. Let (p, q, g) be a

Protocol: (on inputs group G_q, generators g, h)
1. Players generate $\mathsf{RVSS\text{-}data}[a]$ (i.e. perform $\mathsf{Joint\text{-}RVSS}$, Fig.2)
2. Each $P_i \in \mathit{Qual}$ broadcasts his additive shares $a_i, \hat{a}_i$
3. For each $P_i \in \mathit{Qual}$ s.t. $g^{a_i} h^{\hat{a}_i} \neq F_{a_i}(0)$, the players reconstruct P_i's additive share a_i by broadcasting their shares $\alpha_{ij}, \hat{\alpha}_{ij}$ and verifying them with F_{a_i}
4. A public random value a is reconstructed as $a = \sum_{P_i \in \mathit{Qual}} a_i$

Simulation: (on SIM's inputs G_q, g, h and $\sigma = \log_g h$)
1. SIM performs $\mathsf{Joint\text{-}RVSS}$ on the part of the honest players
2. SIM receives random $a^* \in \mathbb{Z}_q$, and for some P_i among the players it controls: SIM broadcasts $a_i^* = a^* - \sum_{P_j \in \mathit{Qual} \setminus \{P_i\}} a_i$ and $\hat{a}_i^*$ s.t. $a_i + \sigma \hat{a}_i = a_i^* + \sigma \hat{a}_i^*$
   For all other players P_j it controls, SIM broadcasts their correct values $a_j, \hat{a}_j$
3. SIM performs Step 3 on the part of the honest players
4. Note that the public random value is reconstructed as a^*

Figure 3: Distributed Coinflip Protocol using $\mathsf{Joint\text{-}RVSS}$

discrete-log instance and assume that the random coins in the proof system are picked in $\mathbb{Z}_q$. Assume that the simulator that exhibits the zero-knowledge property proceeds by first choosing any value for the random coin and then generating the rest of the proof transcript, and that it has zero probability of failure. Three-round ZKPKs of this form exist for, in particular, proving knowledge of discrete logarithm, i.e. $A = \{g^x, x\}$ (e.g. Schnorr's protocol [Sch91]), or knowledge of representations, e.g $A = \{(g, h, g^x h^{\hat{x}}), (x, \hat{x})\}$ (see the work of Brands [Bra99] or Camenisch [Cam98] and the references therein). In a simultaneous proof using a three-round ZKPK, each player P_i proves knowledge of its witnesses w_i for some statement (y_i, w_i) in A in parallel, by executing the steps of the prover as in the original ZKPK protocol, while for the verifier's part, they all use a single common public coin generated with a distributed coinflip protocol. In our protocols, such simultaneous proof is preceeded by h-generation (section 4.2), and the coinflip is implemented with the protocol in Fig.3. This method generalizes to ZKPK protocols with any number of rounds: Every time a public coin is needed, it is picked via a distributed coinflip.

The following lemma is purely technical, but it isolates a convenient property of the simultaneous proof that allow us to concisely argue the security of the protocols that use it as a building block.

Lemma 7 *In the secure channels model, the simultaneous proof protocol has the following two properties: (1) It can be simulated without rewinding* *as long as the simulator has a consistent internal state for every player the adversary corrupts; (2) There is a simulator that can extract all the witnesses from the players controlled by the adversary.*

Proof: Assume we use a 3-round ZKPK proof system with messages m_1, c, m_2 where c is the public coin (we use the notation of Sec. 2). To show property (1), the simulator SIM must conduct the h-generation protocol (section 4.2) to learn $\sigma = \log_g h$. Then, in the simultenous proof, SIM first picks a random coin c. Now, for *some* of the players it controls, SIM might know their witnesses (and all the rest of their internal state). For those players SIM just follows the protocol. For the reminding players, SIM creates a ZKPK transcript $(m_1^{(i)}, c, m_2^{(i)})$ from the coin c, and broadcasts messages $m_1^{(i)}$. Then it simulates an execution of the coinflip protocol that outputs c using the simulator of Fig.3, and broadcasts the corresponding messages $m_2^{(i)}$ for the players it controls. Note that as long as the simulator has a consistent state for the players the adversary corrupts, the simulation is succesful. Note furthermore that SIM never rewinds. As for property (2): If the original 3-round protocol was a proof of knowledge, it must be that once the prover commits itself by its first message, a simulator can extract its witness by gathering its responses on some polynomially-many random coins. The simulator here proceeds in a similar way, by rewinding the protocol to collect the necessary number of responses of the players controlled by the adversary on different coins generated by the distributed coinflip. $\square$

From the lemma above and lemma 5 we immediately get:

Corollary 8 *If the ZKPK proof used in the above simultaneous proof protocol is a committed proof of Fig.1, this protocol can be succesfully simulated without rewinding even if the simulator does not know any witnesses to the statements it reveals for the players it controls.*

Remark: note that we *implement* the honest verifier in this protocol because distributed coinflipping is secure.

4.6 DL key generation

In figure 4, we present a protocol for distributed key generation. As a result of this protocol, a secret/public key pair is generated in an adaptively secure and robust manner. Such a key pair is for a discrete logarithm cryptosystem such as ElGamal [ElG85] or variations.

Lemma 9 *The protocol presented is a secure and robust key generation protocol.*

Proof: Before playing the game with the adversary, we flip a coin to guess whether the adversary is going to try to break the security of the protocol, or its robusteness. With probability $1/2$ our guess is correct, and so we reduce the security and robustness of the protocol to the discrete logarithm problem.

Suppose the adversary wishes to break the security of the protocol. We will prove security by showing that the simulator described in 4 succeeds in furnishing the adversar's view of the execution of the generation of the desired public key y^*. The auxiliary base h was generated such that $\log_g h$ is known to the simulator. First note that the public information created in the simulated run is distributed identically to the public information in the real protocol. Then we note that up to step (4), all players follow the protocol, so any player's internal state can be revealed. After step 4, the only player who has deviated from the protocol is player P_i. Suppose P_i is corrupted. Since step (4) of the protocol dictates that the player erases his additive shares, the simulator can still furnish a consistent internal state for this player. Therefore the view of the adversary is fully simulated.

Now suppose the adversary wishes to break the robustness property of the protocol. Robustness follows through the robustness of Joint-RVSS, and from the fact that a cheating adversary is still required to prove knowledge of discrete logarithm for his contribution y_i. Combining this information, a simulator can infer $\log_g h$. $\square$

4.7 Exponentiating to a shared exponent

In figure 5 we present a protocol for computing m^x on input m, where x is the secret key shared as above. This protocol implements distributed decryption for the ElGamal cryprosystem; it can also be adapted for RSA.

Lemma 10 *The exponentiation protocol is secure and robust.*

Proof: Robustness follows as in the proof of lemma 9.

We prove security by exhibiting a simulator which, when given the information inherited from the simulator for key generation, as well as the desired input m and output o, simulates the view of the adversary in a protocol that results in ouput o.

Protocol: (on inputs group G_q, generators g, h)

1. Players generate RVSS-data[a] (i.e. perform Joint-RVSS, Fig.2)
2. Each $P_i \in \mathcal{Qual}$ computes $y_i = g^{a_i}$, $\hat{y}_i = h^{\hat{a}_i}$.
3. The players execute simultaneous committed proof of knowledge of $\log_g y_i$ and $\log_h \hat{y}_i$.
4. Player P_i erases a_i, $\hat{a}_i$, and all information from which a_i and $\hat{a}_i$ can be reconstructed; broadcasts y_i and $\hat{y}_i$, and opens the committed proof.
5. For each $P_i \in \mathcal{Qual}$, if $y_i \hat{y}_y \neq F_{a_i}(0)$, or if P_i did not carry out a valid proof in step (3), players reconstruct P_i's additive share a_i by broadcasting their shares $\alpha_{ij}, \hat{\alpha}_{ij}$ and verifying them with F_{a_i}
6. The public key is reconstructed as $y = \prod_{P_i \in \mathcal{Qual}} y_i$.
   Player P_i deletes $\alpha_{ji}, \hat{\alpha}_{ji}$, for $j = 1..n$.
   Player P_i computes his polynomial share of the secret $\chi_i = \alpha_i$, $\hat{\chi}_i = \hat{\alpha}_i$.
   Public key verification function F_x is set to be F_a.

Simulation: (on SIM's inputs G_q, g, h, $\sigma = \log_g h$, and the target public key y^*)

1. SIM performs Joint-RVSS on the part of the honest players.
2. Compute the values $y_i, \hat{y}_i$.
3. Simulate the steps of the committed proof.
4. For a server P_i controlled by the simulator, set
   $y_i = y^* / \prod_{P_j \in \mathcal{Qual} \setminus \{P_i\}} y_j$, $\hat{y}_i = F_{a_i}(0)/y_i$.
   For all other players P_j it controls, SIM broadcasts their correct values $y_j, \hat{y}_j$. Open all committed proofs accordingly.
5. SIM performs Step 5 on the part of the honest players
6. Note that the public key is reconstructed as y^*.

Figure 4: DL key generation

Note that the one-time secret a, created in the protocol, ensures that the shares o_i are random values in G_q subject to just one constraint: the Lagrange interpolation in the exponent results in the value o. Thus the information the adversary sees in the simulated run is distributed identically to the one in the real run. $\square$

Protocol: (on inputs group G_q, generators g, h, public key y, secret key share χ_i, $\hat{\chi}_i$, verification function F_x, input m, output o)
1. Players generate $\mathsf{RVSS\text{-}data}_t[a]$ such that $a = 0$ (i.e. perform $\mathsf{Joint\text{-}RVSS}$, Fig.2 for degree $t-1$, and multiply the polynomial by the monomial x).
2. Each $P_i \in \mathit{Qual}$ computes $o_i = m^{\chi_i} g^{\alpha_i}$.
3. The players execute simultaneous committed proof of knowledge and equality of representation of o_i in bases m, g, 1, 1, to that of $F_x(i)$ in bases g, 1, h, 1, to that of $F_a(i)$ in bases 1, g, 1, h.
4. Player P_i erases all information pertaining to a, except the public verification function F_a, and then opens the committed proof.
5. For each $P_i \in \mathit{Qual}$, if P_i did not carry out a valid proof in step (3), other players eliminate P_i from the set Qual.
6. The output o is reconstructed by performing the Lagrange interpolation in the exponent of o_i, for $P_i \in \mathit{Qual}$.

Simulation: (on SIM's inputs G_q, g, h, $\sigma = \log_g h$, public key y, verification function F_x, polynomial ϕ_x such that $F_x(i) = g^{\phi_x(i)}$, input m)
1. SIM performs $\mathsf{Joint\text{-}RVSS}$ on the part of the honest players.
2. —
3. Simulate the steps of the committed proof. If player P_i is corrupted, create random χ_i and set $\hat{\chi}_i = (\phi_x(i) - \chi_i)/\sigma \bmod q$; reveal the rest of the players' state correctly.
4. For all servers P_i controlled by the simulator, set o_i to be random values in G_q such that the Lagrange interpolation will produce o.

Figure 5: Exponentiation

5 Adaptively secure threshold Cramer-Shoup cryptosystem

Recall the Cramer-Shoup [CS98] cryptosystem. The setting is as follows: a group G_q in which the decisional Diffie-Hellman problem is assumed to be hard, and a universal one-way family of hash functions $\mathcal{H} : \{0,1\}^* \to {\mathbb{Z}_q}^*$ are given [BR97, Sho99a]. The secret key consists of five values, a, b, c, d, e, selected from ${\mathbb{Z}_q}^*$ uniformly at random. The public key consists of two random bases, $g_1, g_2 \in G_q$, such that the discrete logarithm that relates them is unknown, and the group elements $C = g_1^a g_2^b$, $D = g_1^c g_2^d$ and $W = g_1^e$. To encrypt a message m from a message space M (M is assumed to have an efficiently computable and invertible mapping into G_q, and so we write $m \in G_q$), Alice chooses $r \in \mathbb{Z}_q{}^*$ uniformly at random, computes $x = g_1^r$, $y = g_2^r$, $w = W^r m$, $\sigma = \mathcal{H}(x, y, w)$, and $v = C^r D^{r\sigma}$. The ciphertext is the 4-tuple (x, y, w, v). And now for decryption, we will use the Canetti-Goldwasser method [CG99]: Bob selects uniformly at random $s \in \mathbb{Z}_q{}^*$ and outputs $w/(x^e(v/v')^s)$, where $v' = x^{a+c\sigma}y^{b+d\sigma}$. Recall that, under the assumption that the decisional Diffie-Hellman problem is hard, the Cramer-Shoup cryptosystem, as well as the Canetti-Goldwasser variant thereof, has been shown to be secure against adaptive chosen ciphertext attack which is the strongest notion of security known for public-key cryptosystems [CS98, Sho99c, CG99].

5.1 Key generation

In figure 6, we present the key generation protocol for the Cramer-Shoup cryptosystem. We assume that the group G_q with a generator g and the universal one-way hash function $\mathcal{H}$ have been generated already. Indeed we may allow one server to set up these parameters and have the others verify that his computation was performed correctly. We also assume that $h \in G_q$ was generated as described in the previous section.

This protocol is a direct generalization of the DL key generation protocol described above.

Lemma 11 *The threshold key generation protocol for the Cramer-Shoup cryptosystem is secure and robust.*

Proof: Robustness follows via a standard argument whereby the existence of an adversary capable of deviating from the protocol without being caught contradicts the discrete logarithm assumption.

Security is proved by exhibiting a simulator in figure 6. The key step in the construction of the simulator is that we generate two auxiliary bases, h_1 and h_2, such that if this is a simulation, the simulator will get to know $\log_{g_1} h_1$ and $\log_{g_2} h_2$. As a result of this and of the committed proof technique, at no step of this protocol will the simulator be committed to a particular player's internal state (see lemma 2 and lemma 5). The additive share of the public key published at the end is non-committing to any current internal state either because it is distributed independently from any non-erased information that the adversary will ever have a chance to see. $\square$

5.2 Decryption

In figure 7, we present the decryption protocol for the Cramer-Shoup cryptosystem. Note that it is a generalization of the shared exponentiation protocol presented in the previous section.

Lemma 12 (Robustness) *The decryption protocol presented is robust.*

Proof: The robustness property follows by a stadard reduction from the discrete logarithm problem. $\square$

Lemma 13 (Correctness) *The decryption protocol outputs the correct decryption of ciphertext (x, y, w, v) under the Cramer-Shoup public key (g_1, g_2, C, D, W).*

Proof: By the robustness property, we may assume that all the players behave as prescribed by the protocol. Let us look at the values O_i:

$$\begin{aligned} O_i &= m_i m_i' \\ &= (v_i/v)^{s_i} g^{r_i s_i} g^{z_i} x^{e_i} g^{-r_i s_i} g^{o_i} g^{-z_i} \\ &= (v_i/v)^{s_i} x^{e_i} g^{r_i s_i + z_i - r_i s_i + o_i - z_i} \\ &= (v_i/v)^{s_i} x^{e_i} g^{o_i} \end{aligned}$$

Since $o(i)$ is a degree $2t$ share of 0, the interpolation of these shares will yield $(v'/v)^{s(0)} u_1^z$, as in Canetti and Goldwasser [CG99]. $\square$

Lemma 14 (Security) *The decryption protocol is secure.*

Proof: We prove security by exhibiting a simulator in figure 7. First note that the public values revealed as a result of the simulation are distributed identically to the public values revealed in a real run of the protocol.

Suppose a server is corrupted before step 4. Then the simulator reveals the values a_i, $\hat{a}_i$, b_i, $\hat{b}_i$, c_i, $\hat{c}_i$, d_i, $\hat{d}_i$, e_i, $\hat{e}_i$ and the correct values for the one-time randomizers generated in step 1. Note that these values are distributed correctly and they comprise the entire secret state of a server.

Suppose a server is corrupted after step 4. Then the simulator reveals the values a_i, $\hat{a}_i$, b_i, $\hat{b}_i$, c_i, $\hat{c}_i$, d_i, $\hat{d}_i$, e_i, $\hat{e}_i$, which are distributed correctly; since the one-time randomizers have been erased, there is no need to reveal anything else. $\square$

5.3 Key refresh

Notice that, using standard techniques [HJJ+97], the above implementation of the threshold Cramer-Shoup cryptosystem can be made *proactive* i.e. secure against mobile attackers who, over time, lose control over some of the servers, but attack new ones.

5.4 Taking the decryption off-line

Note that, as in the Canetti-Goldwasser implementation [CG99], we can precompute and store the randomizers. When a ciphertext needs to be decrypted, a user can talk to each server individually and have each server, using committed proofs, prove to the user that its share of the decryption is valid. By lemma 3, these committed proofs can be executed concurrently. Such a method can tolerate up to $n/3$ corruptions.

ACKNOWLEDGEMENTS

In the first place, I would like to acknowledge the help of Victor Shoup who guided me through this research and supervised my stay at IBM Zürich Research lab. The question of security against the adaptive adversary in the threshold setting arose in a discussion with Victor, as did many of the crucial observations made in this paper. In addition, I would also like to thank Christian Cachin who had to suffer by reading preliminary (new! sensational!) write-ups of my preliminary ideas, and who gave me helpful comments and suggestions. Finally, I acknowledge Stas Jarecki for going over his work [CGJ+99a] with me and helping me situate my contribution with respect to this work; and for his helpful comments about this research. Other people that I am grateful to include, but are not limited to, Ron Rivest, Rosario Gennaro, Klaus Kursawe and Leonid Reyzin.

References

[Abe99] Masayuki Abe. Robust distributed multiplication without interaction. In *Advances in Cryptology—CRYPTO 99*. Springer-Verlag, 1999.

[AWS99] N. Asokan, M. Waidner, and V. Shoup. Optimistic fair exchange of digital signatures. *IEEE Journal of Selected Areas in Communications*, 1999.

[Bea97] Donald Beaver. Plug and play encryption. In *Advances in Cryptology—CRYPTO 97*. Springer-Verlag, 1997.

[BF97] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In *Advances in Cryptology—CRYPTO 97*, pages 425–429. Springer-Verlag, 1997.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Advances in Cryptology—CRYPTO 92*. Springer-Verlag, 1992.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM Symposium on Theory of Computing*, pages 1–10, 1988.

[BH92] D. Beaver and S. Haber. Cryptographic protocols provably secure againts dynamic adversaries. In *Advances in Cryptology—EUROCRYPT 92*. Springer-Verlag, 1992.

[Bon98] Dan Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium*, pages 48–63. Springer-Verlag, 1998.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR97] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: towards making uowhfs practical. In *Advances in Cryptology—CRYPTO 97*, 1997.

[Bra93] Stefan Brands. Decisional Diffie-Hellman problem. *Technical report CWI, CS-R9323*, 1993.

[Bra99] Stefan Brands. Rethinking public-key infrastructures and digital certificates–building in privacy. *Ph.D. dissertation, Technical University of Eindhoven*, 1999.

[Cam98] Jan Camenisch. Group signature schemes and payment systems based on the discrete logarithm problem. *ETH Series in Information Security and Cryptography, vol.2*, 1998.

[Can98] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Theory of Cryptography Library, http://philby.ucsd.edu/cryptolib/1998.html*, 1998.

[CCD88] David Chaum, Claude Crepeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 11–19, 1988.

[CD98] Ronald Cramer and Ivan Damgård. Zero-knowledge proof for finite field arithmetics, or: Can zero-knowledge be for free. In *Advances in Cryptology—CRYPTO 98*, pages 424–441. Springer-Verlag, 1998.

[CDD+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology—EUROCRYPT 99*, pages 311–326. Springer-Verlag, 1999.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 639–648, 1996.

[CFIJ99] Giovanni Di Crescenzo, Niels Ferguson, Russell Impagliazzo, and Markus Jakobsson. How to forget a secret. In *Proceedings of STACS'99*, 1999.

[CG99] Ran Canetti and Shafi Goldwasser. An efficient threshold public key cryptosystem secure against adaptice chosen ciphertext attack. In *Advances in Cryptology—EUROCRYPT 99*, pages 90–106. Springer-Verlag, 1999.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. Random oracle methodology, revisited. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 209–218, 1998.

[CGJ+99a] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Advances in Cryptology—CRYPTO 99*. Springer-Verlag, 1999.

[CGJ+99b] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. *http://theory.lcs.mit.edu/˜cis/cis-publications.html*, 1999.

[CS98] Ronald Cramer and Victor Shoup. A practical public-key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology—CRYPTO 98*. Springer-Verlag, 1998.

[Des87] Yvo Desmedt. Society and group oriented cryptography. In *Advances in Cryptology—CRYPTO 87*. Springer-Verlag, 1987.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology—CRYPTO 89*, pages 307–315. Springer-Verlag, 1989.

[DNRS99] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry Stockmeyer. Magic functions. In *40th IEEE Symp. on Foundations of Comp. Science*, 1999.

[ElG85] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FGMY97] Yair Frankel, Peter Gemmell, Philip MacKenzie, and Moti Yung. Optimal-resilience proactive public-key cryptosystems. In *Proc. 38th IEEE Symp. on Foundations of Comp. Science*, 1997.

[FGY96] Yair Frankel, Peter Gemmell, and Moti Yung. Witness-based cryptographic program checking and robust function sharing. In *Proc. 28th ACM Symp. on Theory of Computing*, 1996.

[FHM98] Matthias Fitzi, Martin Hirt, and Ueli Maurer. Trading correctness for privacy in unconditional multi-party computation. In *Advances in Cryptology—CRYPTO 98*. Springer-Verlag, 1998.

[FMY98] Yair Frankel, Philip MacKenzie, and Moti Yung. Robust efficient distributed rsa-key generation. In *Proc. 30th ACM Symp. on Theory of Computing*, 1998.

[FMY99a] Yair Frankel, Philip MacKenzie, and Moti Yung. Adaptively-secure distributed threshold public key systems. In *Proceedings of ESA 99*, 1999.

[FMY99b] Yair Frankel, Philip MacKenzie, and Moti Yung. Adaptively-secure optimal-resilience proactive RSA. In *Advances in Cryptology—ASIACRYPT 99*. Springer-Verlag, 1999.

[GB96] Shafi Goldwasser and Mihir Bellare. Lecture notes in cryptography. *ftp://theory.lcs.mit.edu/pub/classes/6.875/crypto-notes.ps*, 1996.

[GJKR96a] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of rsa functions. In *Advances in Cryptology—CRYPTO 96*. Springer-Verlag, 1996.

[GJKR96b] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signature. In *Advances in Cryptology—EUROCRYPT 96*. Springer-Verlag, 1996.

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete–log based cryptosystems. In *Advances in Cryptology—EUROCRYPT 99*, pages 295–310, 1999.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[Gol95] Oded Goldreich. Foundations of cryptography: Fragments of a book. *http://theory.lcs.mit.edu/˜oded*, 1995.

[Gol98] Oded Goldreich. Secure multi-party computation. *http://theory.lcs.mit.edu/˜oded*, 1998.

[HJJ+97] Amir Herzberg, Markus Jakobsson, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In *4th ACM Conf. on Comp. and Comm. Security*, pages 100–110, 1997.

[HJKY95] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In *Advances in Cryptology—CRYPTO 95*, pages 339–352. Springer-Verlag, 1995.

[JL00a] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: introducing concurrency, removing erasure. *EUROCRYPT2000*, 2000.

[JL00b] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography without erasures. *Theory of Cryptography Library*, 2000.

[Lys00] Anna Lysyanskaya. Threshold cryptography secure against the adaptive adversary, concurrently. *Theory of Cryptography Library*, 2000.

[Ped91a] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—CRYPTO 91*, pages 129–140. Springer-Verlag, 1991.

[Ped91b] T. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT 91*, pages 522–526, 1991.

[Rab98] Tal Rabin. A simplified approach to threshold and proactive RSA. In *Advances in Cryptology—CRYPTO 98*, pages 89–104. Springer-Verlag, 1998.

[RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st ACM Symp. on Theory of Computing*, 1989.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[SG98] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Advances in Cryptology—EUROCRYPT 98*. Springer-Verlag, 1998.

[Sho99a] Victor Shoup. A composition theorem for universal one-way hash functions. *IBM Research Report RZ3147*, 1999.

[Sho99b] Victor Shoup. Practical threshold signatures. *IBM Research Report RZ3121*, 1999.

[Sho99c] Victor Shoup. Why chosen ciphertext security matters. *IBM Research Report RZ3076*, 1999.

[Sip97] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[SS94] Michael Sipser and Dan Spielman. Expander codes. In *Proc. 35th IEEE Symp. on Foundations of Comp. Science*, 1994.

[Wol99] Stefan Wolf. Information-theoretically and computationally secure key agreement in cryptography. *Ph.D. thesis, ETH Zürich*, 1999.

[]

Input: G_q, g, h, $\mathcal{H}$
Goal: Generate the Cramer-Shoup public key (g_1, g_2, C, D, W).

1. Run the joint coinflip protocol and generate random bases g_1, g_2, h_1, h_2.
2. Run Joint-RVSS five times in parallel and obtain $\mathsf{RVSS\text{-}data}_{t,g_1,h_1}[a,c,e]$ and $\mathsf{RVSS\text{-}data}_{t,g_2,h_2}[b,d]$.
3. P_i performs, in parallel, committed simultaneous proofs of knowledge of repr. in bases g_1, g_2 of values $C_i = g_1^{a_i} g_2^{b_i}$, $D_i = g_1^{c_i} g_2^{d_i}$ and $W_i = g_1^{e_i}$;
   and repr. in bases h_1, h_2 of values $\hat{C}_i = h_1^{\hat{a}_i} h_2^{\hat{b}_i}$, $\hat{D}_i = h_1^{\hat{c}_i} h_2^{\hat{d}_i}$, and $\hat{W}_i = g_1^{\hat{e}_i}$;
   P_i erases f_{a_i}, f_{b_i}, f_{c_i}, f_{d_i}, f_{e_i} and $f_{\hat{a}_i}$, $f_{\hat{b}_i}$, $f_{\hat{c}_i}$, $f_{\hat{d}_i}$, $f_{\hat{e}_i}$;
   P_i opens the committed proofs.
4. Verify (1) validity of other players' proofs;
   and (2) for all $P_k \in \mathit{Qual}$, $C_k \hat{C}_k = F_{a_k}(0) F_{b_k}(0)$, $D_k \hat{D}_k = F_{c_k}(0) F_{d_k}(0)$,
   and $W_k \hat{W}_k = F_{e_k}(0)$.
   For any player who failed the test, reconstruct all his secrets using backup information stored in $\mathsf{RVSS\text{-}data}[a,b,c,d,e]$.
5. Compute the public key:
   $C = \prod_{P_i \in \mathit{Qual}} C_i$, $D = \prod_{P_i \in \mathit{Qual}} D_i$ and $W = \prod_{P_i \in \mathit{Qual}} W_i$.

Simulation: (on SIM's inputs G_q, g, h, $\sigma = \log_g h$, public key $(g_1, g_2, C^*, D^*, W^*)$)
1. SIM runs the simulator for the coinflip protocol to obtain g_1, g_2, and the auxiliary bases h_1, h_2 such that $\sigma_i = \log_{g_i} h_i$, $i = 1, 2$, is known to SIM.
2. Similarly to SIM for the DL key generation protocol, perform Joint-RVSS on behalf of the uncorrupted players.
3. Simulate the steps of the committed proof. For a server P_i controlled by the simulator, set
   $C_i = C^* / \prod_{P_j \in \mathit{Qual} \setminus \{P_i\}} C_j$, $\hat{C}_i = F_{a_i}(0) F_{b_i}(0) / C_i$; $D_i = D^* / \prod_{P_j \in \mathit{Qual} \setminus \{P_i\}} D_j$, $\hat{D}_i = F_{c_i}(0) F_{d_i}(0) / D_i$; $C_i = W^* / \prod_{P_j \in \mathit{Qual} \setminus \{P_i\}} W_j$, $\hat{W}_i = F_{e_i}(0)/W_i$.
   For all other players P_j it controls, SIM broadcasts their correct values C_j,$\hat{C}_j$; D_j,$\hat{D}_j$; W_j,$\hat{W}_j$. Open all committed proofs accordingly.
4. SIM performs Step 4 on the part of the honest players.
5. Note that the public key is reconstructed as $(g_1, g_2, C^*, D^*, W^*)$.

Figure 6: Key generation for the Cramer-Shoup cryptosystem

Input: Values obtained from the key generation protocol.
Goal: Decrypt ciphertext (x, y, w, v).
Notation: In this protocol, indexed Latin letters (e.g. a_i) denote *polynomial* shares of the corresponding values. (Unlike the rest of this paper where they denote additive shares.)

1. Run Joint-RVSS five times in parallel and obtain $\mathsf{RVSS\text{-}data}_{t,g,h}[s, r, p]$ and $\mathsf{RVSS\text{-}data}_{2t,g,h}[o, z, u]$, where $\mathsf{RVSS\text{-}data}[o]$ is a sharing of 0.
2. P_i computes the following values:
   (a) $l_i = x^{a_i+c_i\sigma}y^{b_i+d_i\sigma}g^{r_i} = v_i g^{r_i}$, where $v_i = x^{a_i+c_i\sigma}y^{b_i+d_i\sigma}$.
   (b) $l'_i = g^{r_i}h^{p_i}$.
   (c) $l''_i = g^{r_i s_i}h^{p_i s_i+u_i} = (l'_i)^{s_i}h^{u_i}$.
   (d) $m_i = (l_i/v)^{s_i}g^{z_i} = (v_i/v)^{s_i}g^{r_i s_i}g^{z_i}$.
   (e) $m'_i = x^{e_i}g^{-r_i s_i}g^{o_i}g^{-z_i}$.
3. Prove in committed simultaneous form:
   (a) Eq. of repr. of l_i, $F_a(i)$, $F_b(i)$, $F_c(i)$, $F_d(i)$, $F_r(i)$ in bases $(x, x^\sigma, y, y^\sigma, g, 1, 1, 1, 1, 1)$, $(g_1, 1, 1, 1, 1, h_1, 1, 1, 1, 1)$, $(1, 1, g_2, 1, 1, 1, 1, h_2, 1, 1)$, $(1, g_1, 1, 1, 1, 1, h_1, 1, 1, 1)$, $(1, 1, 1, g_2, 1, 1, 1, 1, h_2, 1)$, $(1, 1, 1, 1, g, 1, 1, 1, 1, h)$, correspondingly.
   (b) Eq. of repr. of l'_i, $F_r(i)$, $F_p(i)$ in bases $(g, 1, h, 1)$, $(g, h, 1, 1)$, $(1, 1, g, h)$.
   (c) Eq. of repr. of l''_i, $F_s(i)$, $F_u(i)$ in bases $(l'_i, h, 1, 1)$, $(g, 1, h, 1)$, $(1, g, 1, h)$.
   (d) Eq. of repr. of m_i,$F_s(i)$, $F_z(i)$ in bases $((l_i/v), g, 1, 1)$, $(g, 1, h, 1)$, $(1, g, 1, h)$.
   (e) Eq. of repr. of m'_i, $F_e(i)$, l''_i, $F_o(i)$, $F_z(i)$ in bases $(x, g^{-1}, g, g^{-1}, 1, 1, 1, 1)$, $(g_1, 1, 1, 1, h_1, 1, 1, 1)$, $(1, g, 1, 1, 1, h, 1, 1)$, $(1, 1, g, 1, 1, 1, h, 1)$, $(1, 1, 1, g, 1, 1, 1, h)$.
4. Erase the one-time secrets generated in step 1.
5. Open the committed proofs and reveal l_i, l'_i, l''_i, m_i, and m'_i.
6. Verify the committed proofs of other players.
7. Set a players output share $O_i = m_i m'_i$. Determine the output O by Lagrange interpolation in the exponent; the resulting decryption is w/O.

Figure 7: Decryption for the Cramer-Shoup cryptosystem

Input to simulation: G_q, g, h, $\sigma = \log_g h$, public key $(g_1, g_2, C^*, D^*, W^*)$, ciphertext (x, y, w, v), target decryption O^*, verification functions F_a, F_b, F_c, F_d, F_e, polynomials ϕ_a, ϕ_b, ϕ_c, ϕ_d, ϕ_e, such that $\forall i \in \mathbb{Z}_q$, $F_a(i) = g_1^{\phi_a(i)}$, $F_b(i) = g_2^{\phi_b(i)}$, $F_c(i) = g_1^{\phi_c(i)}$, $F_d(i) = g_2^{\phi_d(i)}$, $F_e(i) = g_1^{\phi_e(i)}$.

1. Run Joint-RVSS on behalf of the uncorrupted players; obtain polynomials ϕ_s, ϕ_r, ϕ_p, ϕ_o, ϕ_z, ϕ_u such that $F_s(i) = g_1^{\phi_s(i)}$, $F_r(i) = g_1^{\phi_r(i)}$, $F_p(i) = g_1^{\phi_p(i)}$, $F_o(i) = g_2^{\phi_o(i)}$, $F_z(i) = g_2^{\phi_z(i)}$, $F_u(i) = g_2^{\phi_u(i)}$.
2. For a player P_i SIM controls, generate random values a_i, b_i, c_i, d_i, e_i, and set $\hat{a}_i = (\phi_a(i) - a_i)/\sigma_1$, $\hat{b}_i = (\phi_b(i) - b_i)/\sigma_2$, $\hat{c}_i = (\phi_c(i) - c_i)/\sigma_1$, $\hat{d}_i = (\phi_d(i) - d_i)/\sigma_2$, $\hat{e}_i = (\phi_e(i) - e_i)/\sigma_1$. Then compute l_i, l'_i, l''_i, m_i and m'_i as prescribed.
3. Perform the steps of the committed proofs.
4. —
5. For the players still controlled by SIM, open the committed proofs as follows: l_i, l'_i, l''_i, m_i and m'_i are random elements of G_q with the constraints that (1) they interpolate to polynomials of the right degrees and (2) the products $m_i m'_i$ interpolate to the target decryption O^*.
6. Verify the committed proofs of other players.
7. We have guaranteed that the resulting decryption is w/O^*.

Figure 8: Simulation of the decryption protocol