Adaptive Security for Threshold Cryptosystems

Ran Canetti[1], Rosario Gennaro[1], Stanisław Jarecki[2],
Hugo Krawczyk[3], and Tal Rabin[1]

[1] IBM T.J.Watson Research Center
PO Box 704, Yorktown Heights, NY 10598, USA
{canetti,rosario,talr}@watson.ibm.com
[2] MIT Laboratory for Computer Science
545 Tech Square, Cambridge, MA 02139, USA
stasio@theory.lcs.mit.edu
[3] Department of Electrical Engineering, Technion
Haifa 32000, Israel
and
IBM T.J. Watson Research Center, New York, USA
hugo@ee.technion.ac.il

Abstract. We present *adaptively-secure* efficient solutions to several central problems in the area of threshold cryptography. We prove these solutions to withstand *adaptive attackers* that choose parties for corruption at any time during the run of the protocol. In contrast, all previously known efficient protocols for these problems were proven secure only against less realistic *static adversaries* that choose and fix the subset of corrupted parties before the start of the protocol run.
Specifically, we provide adaptively-secure solutions for distributed key generation in discrete-log based cryptosystems, and for the problem of distributed generation of DSS signatures (threshold DSS). We also show how to transform existent static solutions for threshold RSA and proactive schemes to withstand the stronger adaptive attackers. In doing so, we introduce several techniques for the design and analysis of adaptively-secure protocols that may well find further applications.

1 Introduction

Distributed cryptography has received a lot of attention in modern cryptographic research. It covers a variety of areas and applications, from the study of secret-sharing schemes, to the distributed computation of general functions using secure multi-party protocols, to the design and analysis of specific threshold cryptosystems. Two main goals that motivate this research area are: (i) provide security to applications that are inherently distributed, namely, several parties are trying to accomplish some common task (e.g., secure elections, auctions, games) in the presence of an attacker, and (ii) avoid single points-of-failure in a security system by distributing the crucial security resources (e.g. sharing the ability to generate signatures). In both cases the underlying assumption is that an attacker can penetrate and control a portion of the parties in the system but not all of them.

Coming up with correct protocols for meeting the above goals has proven to be a challenging task; trying to design protocols that are practical as well

Michael Wiener (Ed.): CRYPTO'99, LNCS 1666, pp. 98–116, 1999.

as fully-analyzed is even more challenging. One inherent difficulty in the analysis of cryptographic systems, in general, is the need to define a mathematical model that is strong enough to capture realistic security threats and attacker's capabilities, and which, at the same time, allows to prove the security of sound solutions. Due to the complexity of building and reasoning about distributed protocols, this difficulty is even greater in the area of distributed cryptography. In this case a large variety of security models have been proposed. This is not just due to "philosophical disagreements" on what the best way to model reasonable attackers is, but it is also influenced by our ability (or lack of it) to prove security of protocols under strong adversarial models.

One major distinction between security models for distributed protocols is whether the attacker is *static* or *adaptive* (the latter is also called "dynamic"). In both cases the attacker is allowed to corrupt any subset of parties up to some size (or threshold) as specified by the security model. However, in the case of an adaptive adversary the decision of which parties to corrupt can be made at any time during the run of the protocol and, in particular, it can be based on the information gathered by the attacker during this run. In contrast, in the static case the attacker must choose its victims independently of the information it learns during the protocol. Therefore, the subset of corrupted parties can be seen as chosen and fixed by the attacker before the start of the protocol's run.

While the adaptive attacker model appears to better capture real threats, the bulk of published works on distributed cryptography assumes a static adversary. This is due to the difficulties encountered when trying to design and prove protocols resistant to adaptive adversaries. Still, general constructions have been shown in the adaptive-adversary model for secure distributed evaluation of any polynomial-time computable function (see below). However, these general results do not provide sufficiently efficient and practical solutions. Until now, no efficient adaptive solutions for threshold cryptosystems were known.

Our Contribution. The main contribution of this paper is in providing concrete, fully-specified, fully-analyzed solutions to some of the central problems in threshold cryptography, and proving their security in the adaptive adversary model. Our solutions add little overhead relative to existing solutions for the same problems in the static-adversary model. They are also constant-round; namely, the number of rounds of communication is fixed and independent of the number of parties in the system, the input length, or the security parameter. Thus we believe that our protocols can be of real use in emerging threshold cryptography applications. Very importantly, we provide full analysis and proofs for our solutions. This is essential in an area where simple intuition is usually highly misleading and likely to produce flawed protocols.

In addition, our work introduces general new techniques for the design and analysis of protocols in the adaptive-adversary model. In section 2 we give an overview of these techniques to aid the understanding of the protocols and the proofs given in the following sections. We also hope that these techniques will be applicable to the design of adaptively-secure solutions to other problems.

In section 3 we start the presentation of our protocols with an adaptively-secure solution for the distributed generation of keys for DSS, El Gamal, and other discrete-log based public-key systems (for signatures and encryption). Such a protocol is not only needed for generating keys in a distributed way without a trusted party, but it is also a crucial component of many other cryptographic protocols. We illustrate this in our own work by using it for distributed generation of the $r = g^k$ part in DSS signatures.

In section 4 we present a threshold DSS protocol for the shared generation of DSS signatures that withstands adaptive attackers. We first show a simplified protocol where the attacker can control up to $t < n/4$ players. This protocols helps in highlighting and understanding many of our new and basic techniques. Next we describe an optimal-resilience, $t < n/2$, threshold DSS protocol.

In section 5 we show how our techniques can be used to achieve adaptively-secure distributed protocols for other public-key systems. We show how to modify existing threshold RSA protocols (proven in the static model) to obtain adaptively-secure threshold RSA. Here also we achieve optimal-resiliency and constant-round protocols. Similarly, our techniques allow for "upgrading" existent proactive discrete-log based threshold schemes from the static-adversary model to the adaptive one.

Related Work. Our work builds directly on previous protocols that were secure only in the static-adversary model, particularly on [Ped91b,GJKR99] for the distributed key generation and [GJKR96] for the threshold DSS signatures. We modify and strengthen these protocols to achieve adaptive security. Similarly, our solution to adaptive threshold RSA is based on the previous work of [Rab98].

As said before, if one is not concerned with the practicality of the solutions then general protocols for secure distributed computation of polynomial-time functions in the presence of adaptive adversaries are known. This was shown in [BGW88,CCD88] assuming (ideal) private channels. Later, [BH92] showed that with the help of standard encryption and careful erasure of keys one can translate these protocols into the model of public tappable channels. Recently, [CFGN96] showed how to do this translation without recurring to erasures but at the expense of a significant added complexity. Other recent work on the adaptive-adversary model includes [Can98,CDD+99]. Also, independently from our work, adaptively-secure distributed cryptosystems have been recently studied in [FMY].

2 Technical Overview: Road-Map to Adaptive Security

This section provides an overview of some basic technical elements in our work. It is intended as a high-level introduction to some of the issues that underly the protocol design and proofs presented in this paper (without getting into a detailed description of the protocols themselves). We point out to some general aspects of our design and proof methodology, focusing on the elements that are essential to the treatment of the adaptive-adversary model in general, and to the understanding of our new techniques. For simplicity and concreteness, our

presentation focuses mainly on threshold signature schemes which are threshold solutions to some existing centralized scheme like DSS or RSA. However, most of the issues we raise are applicable to other threshold functions and cryptosystems.

Threshold Signature Schemes. A threshold signature scheme consists of a distributed key generation protocol, a distributed signature generation protocol and a centralized verification algorithm. The signing servers first run the key generation, and obtain their private key shares and the global public key. Next, whenever a message needs to be signed, the servers invoke the signature generation protocol. The definition of secure threshold signature scheme makes two requirements. The first is *unforgeability*, which says, roughly, that, even after interacting with the signing parties in the initial generation of the distributed key and then in the signing protocol invoked on adaptively chosen messages, the adversary should be unable (except with negligible probability) to come up with a message that was not signed by the servers together with a valid signature. The second requirement is *robustness*, which says that whenever the servers wish to sign a message, a valid signature should be generated. Our model and definitions are a natural adaptation to the adaptive-adversary model of the definitions for threshold signatures found in [GJKR96], which in turn are based on the unforgeability notions of [GMR88].

Proofs by Reduction and the Central Role of Simulations. We use the usual "reductionist" approach for the proofs of our protocols. Namely, given an adversary $\mathcal{A}$ that forges signatures in the distributed setting, we construct a forger $\mathcal{F}$ that forges signatures of the underlying centralized signature scheme. Thus, under the assumption that the centralized scheme is secure, the threshold signature scheme must also be secure. A key ingredient in the reduction is a "simulation" of the view of the adversary $\mathcal{A}$ in its run against the distributed protocol. That is, the forger $\mathcal{F}$ builds a virtual distributed environment where the instructions of $\mathcal{A}$ are carried out. Typically, first $\mathcal{F}$ has to simulate to $\mathcal{A}$ an execution of the distributed key generation that results in the same public key against which $\mathcal{F}$ stages a forgery. Then $\mathcal{A}$ will successively invoke the distributed signature protocol on messages of his choice, and $\mathcal{F}$, having access to a signing oracle of the underlying centralized signature scheme, has to simulate to $\mathcal{A}$ its view of an execution of the distributed signature protocol on these messages. Eventually, if $\mathcal{A}$ outputs a forgery in the imitated environment, $\mathcal{F}$ will output a forgery against the centralized signature scheme. Two crucial steps in the analysis of this forger are: (1) Demonstrate that the adversary's view of the simulated interaction is indistinguishable from its view of a real interaction with parties running the threshold scheme. (2) Demonstrate that the forger can translate a successful forgery by the adversary (in the simulated run) to a successful forgery of the centralized signature scheme.

The first step is the technical core of our proofs. Furthermore, to carry out the second step we need simulators that are able to generate views that are indistinguishable from the view of the adversary *under a given conditioning.* More specifically, $\mathcal{F}$ has to simulate a run of the distributed key generation that arrives at a *given* public key of the underlying centralized scheme, and it has

to simulate a run of the distributed signature scheme that arrives at a *given* signature output by the signing oracle of the centralized scheme. We refer to this as the problem of *hitting* a particular value in the simulation.

Problems with Simulating an Adaptive Adversary. The above proof technique is also used in the case of static adversaries. However, an adaptive adversary can corrupt any player at any time (as long as not too many parties are corrupted) and at that point the simulator needs to be able to provide the attacker with the current internal state of the broken party. In particular, this information must be consistent with the information previously seen by the attacker. Providing this information is typically the main difficulty in proving adaptive security of protocols.

We demonstrate this difficulty with a simplified example adapted from our protocols: Assume that the protocol specifies that each server P_i chooses a random exponent x_i and makes public (broadcasts) g^{x_i}, where g is a known group generator. Next, the adversary sees $g^{x_1}...g^{x_n}$, corrupts, say, 1/3 of the servers, and expects to see the secret exponents x_i of the corrupted servers being consistent with g^{x_i}. Since the simulator cannot predict which servers will be corrupted, the only known way to carry out the simulation is to make sure that the simulator knows in advance the secret values x_i of *all* the servers, even though the adversary corrupts only a fraction of them. However, this is not possible in our protocols where the x_i's encode a secret quantity x (such as a signing key) unknown to the simulator. Note that trying to guess which servers will be corrupted does not help the simulation: there is an exponential number of such sets. (In contrast, the simulation of such a protocol *is* possible in the case of a static attacker where the simulator knows in advance the set of corrupted players.)

Erasures. One way to get around the above problem in the adaptive-adversary model is to specify in the protocol that the servers *erase* the private values x_i before the values g^{x_i} are broadcasted. Now, when corrupting P_i, this information is not available to the adversary in the real run of the protocol and therefore there is no need to provide it in the simulation. However, this technique can only be applied when the protocol no longer needs x_i. A careful use of erasures is at the core of the design of our protocols. In some cases, this requires that information that could have been useful for the continuation of the protocol be erased. Two examples of crucial erasures in our protocols are the erasure of some of the verification information kept by Pedersen's VSS protocol [Ped91a] (which we compensate for with the use zero-knowledge proofs – see below), and the erasure of all temporary information generated during each execution of the signature protocol. Furthermore, erasures simplify the task of implementing private channels with conventional encryption in the adaptive model (see below).[1]

[1] Successful erasure of data is not a trivial task; one needs to make sure that the data, and all its back-ups, are carefully overwritten. In our setting, we trust *uncorrupted* parties to properly erase data whenever required by the protocol. (See [CFGN96,Can98] for further discussion on the issue of data erasures.)

Rewinding the Adversary. Another useful technique for getting around the problem of having to present the attacker with information not available to the simulator is rewinding. This is a well-known technique for proving zero-knowledge (and other) protocols. In its essence it allows a simulator that is "in trouble" with its simulation to rewind the adversary's state to a previous computation state, and restart the computation from there. At this point the simulation will try some other random choices in the computation hoping that it will not end in another bad position as before. Thus, in the case where the chances to get "stuck" again are not too large then rewinding is a very useful technique. In a second try, after the simulator makes some different choices, the adversary will hopefully not hit an "inconsistent" situation again.

Note that rewinding is a proof technique, not a protocol action. Yet, one has to design the protocol in ways that make rewinding a useful tool. As in the case of erasures, correct use of rewinding requires care. Following is an important instance where an improper use of rewinding can render the whole simulation useless.[2] Assume that the distributed adversary $\mathcal{A}$ asks for signatures on a sequence of messages $m_1, m_2, \ldots$. For each message m_i, in order to simulate the signature protocol, the forger $\mathcal{F}$ must ask the signing oracle of the underlying centralized signature scheme for a signature on m_i. Assume now that during the k^{th} signature protocol (while simulating the distributed signing of message m_k) the simulator gets "stuck" and needs to rewind the adversary back to a *previous* signature protocol, say the j^{th} one for $j < k$. Now the adversary is rewinded back to m_j, the subsequent view is different, so with all likelihood the adversary will start asking for signatures on *different* messages. In other words the sequence of messages is now $m_1, \ldots, m_j, m'_{j+1}, \ldots, m'_k, \ldots$. However the sequence of messages asked by the forger $\mathcal{F}$ to the signature oracle is $m_1, \ldots, m_j, m_{j+1}, \ldots, m_k, m'_{j+1}, \ldots, m'_k$, i.e. the forger has now asked *more* messages than the adversary (indeed since the adversary was rewinded he has no recollection of asking the messages $m_{j+1}, \ldots, m_k$). This means that the adversary $\mathcal{A}$ may output one of those messages as a successful forgery, but such event will not count as a success for the forger $\mathcal{F}$.

It is important then to confine rewinding of the adversary *inside* a simulation of a single run of the signature protocol. One of the tools that we use to ensure that our protocols are simulatable in such a way is the erasure of local temporary information generated by our protocols.

The Single-Inconsistent-Player Technique. Another concern that needs to be addressed is making sure that rewindings do not take place too often (otherwise the simulator may not run in polynomial time). Very roughly, we guarantee this property as follows. We make sure that the simulator can, at any point of the simulated run, present the attacker with a correct internal state (i.e. state that is consistent with the attacker's view) for all honest players except, maybe,

[2] We remark that the rewinding technique may also cause difficulties when the signature protocol is composed with other protocols, and in particular when several copies of the signature protocol are allowed to run *concurrently*. We leave these issues out of the scope of this work. See more details in [DNS98,KPR98].

for *one server*. The identity of this server is chosen at random. Moreover, the view of the attacker is independent from this choice. This guarantees that the inconsistent server is corrupted (and then the simulation is rewinded) with probability at most one half (this probability is given by the ratio t/n of corrupted players). Thus, the expected number of rewindings is at most one.

Zero-Knowledge to the Rescue, and Efficiently. Another tool used in our protocols are zero-knowledge proofs [GMR89], and more specifically zero-knowledge proofs of knowledge. Useful as they are, zero-knowledge proofs may add significant complexity to the protocols, degrade performance, and increase communication. We show how to make intensive use of zero-knowledge proofs with significant savings in complexity. Specifically, we show how to achieve the effect of $O(n^2)$ zero-knowledge proofs of knowledge (where each of the n players proves something to each of the other players) in a *single* 3-move *honest verifier* zero-knowledge proof. This is done by implementing the honest verifier using a distributed generation of a challenge by all players.[3] We implement this technique for Schnorr's proof of possession of discrete-log [Sch91]; the same technique can be used for other zero-knowledge protocols as well [CD98].

Maintaining Private Channels in the Adaptive Model. An important observation about the design of our protocols is that we specify them using the abstraction of "private channels" between each pair of parties. This is a usual simplifying approach in the design of cryptographic protocols: The underlying assumption is that these private channels can be implemented via encryption. In the case of adaptive security, however, this simple paradigm needs to be re-examined. Indeed, a straightforward replacement of private channels with encryption could ruin the adaptive security of our protocols. Fortunately, in settings where data erasures are acceptable (and in particular in our setting) a very simple technique exists [BH92] for solving this problem. It involves local refreshment of (symmetric) encryption keys by each party, using a simple pseudorandom generator and without need for interaction between the parties. This adds virtually no overhead to the protocols beyond the cost of symmetric encryption itself.

3 Adaptively-Secure Distributed Key Generation

A basic component of threshold cryptosystems based on the difficulty of computing discrete logarithms is the shared generation of a secret x for which the value g^x is made public. Not only is this needed to generate a shared key without a trusted dealer but it is also a sub-module of other protocols, e.g. as used in our own threshold DSS scheme for generating $r = g^{k^{-1}}$ (see Sec. 4). We call this module a "Distributed Key Generation" (DKG) protocol. The primitive of

[3] Recall that 3-move zero-knowledge proofs cannot exist for *cheating* verifiers if the underlying problem is not in BPP [GK96,IS93]. Thus, the distributed nature of the verifier in our implementation is essential for "forcing honesty". In particular, our simulation of these proofs does not require rewinding at all.

Distributed Key Generation for Discrete–Log Based Cryptosystems is defined in [GJKR99], yet the solution provided in that paper is proven secure only against a non-adaptive adversary. Briefly stating, a DKG protocol is performed by n players $P_1, ..., P_n$ on public input (p, q, g) where p and q are large primes, q divides $p-1$, and g is an element of order q in Z_p^*. DKG generates a Shamir secret sharing of a uniformly distributed random secret key $x \in Z_q$, and makes public the value $y = g^x \bmod p$. At the end of the protocol each player has a private output x_i, called a *share* of x. The protocol is secure with threshold (t, n) if in the presence of an adversary who corrupts at most t players, the protocol generates the desired outputs and does not reveal any information about x except for what is implied by the public value $g^x \bmod p$.

To ensure the ability to use the DKG protocol as a module in a larger, adaptively secure, threshold scheme (like the DSS-ts signature scheme of Section 4) we add to the requirements of [GJKR99] that at the end of the protocol each party must *erase* all the generated internal data pertaining to this execution of the protocol except of course for its private output x_i. This requirement ensures that the simulator of the threshold scheme within which DKG is used as a module (e.g. DSS-ts) is able to produce the internal state of a player whom the adversary corrupts *after* the execution of the DKG module is completed.

Distributed Key Generation Protocol. We present a distributed key generation protocol DKG with resilience $t < n/3$, which is simple to explain and already contains the design and analysis ideas in our work. (See below for the modifications required to achieve optimal resilience $t < n/2$.) DKG is based on the distributed key generation protocol of [GJKR99] which is proven secure only against a non-adaptive adversary. (Some of the changes made to this protocol in order to achieve adaptive security are discussed below.)

Protocol DKG presented in Fig.1, starts with inputs (p, q, g, h) where (p, q, g) is a discrete–log instance and h is a random element in the subgroup of Z_p^* generated by g. When DKG is executed on inputs (p, q, g) only, a random h must first be publicly generated as follows: If q^2 does not divide $p-1$, the players generate a random $r \in Z_p^*$ via a collective coin-flipping protocol [BGW88] and take $h = r^{(p-1)/q} \bmod p$.[4] The protocol proceeds as follows: The first part of generating x is achieved by having each player commit to a random value z_i via a Pedersen's VSS [Ped91a,Ped91b,GJKR99]. These commitments are verified by the other players and the set of parties passing verification is denoted by *QUAL*. Then the shared secret x is set (implicitly) to $x = \sum_{i \in QUAL} z_i \bmod q$. We denote this subprotocol by Joint-RVSS (see Figure 2).[5] In addition to enabling the generation of a random, uniformly distributed value x, Joint-RVSS has the side effect of having each player P_i broadcast an information-theoretically private commitment to z_i of the form $C_{i0} = g^{z_i} h^{f_i'(0)} \bmod p$, where f_i' is a random-

[4] We chose to write DKG with h as an input so that it could be invoked as a module by the DSS-ts scheme of Section 4 without generating h each time.

[5] See [GJKR99] for an analysis of Joint-RVSS in the non-adaptive adversarial model. The same analysis applies to the adaptive model treated here.

Input: Parameters (p, q, g), and h an element in the subgroup generated by g.
Public Output: y the public key
Secret Output of P_i: x_i the share of the random secret x
(all other secret outputs are erased)
Other Public Output: Public commitments.
Generating x:
Players execute Joint-RVSS(t, n, t)

1. Player P_i gets the following secret outputs of Joint-RVSS:
   – x_i, x'_i his share of the secret and the associated random value (resp.)
   – $f_i(z), f'_i(z)$ polynomials he used to share his contribution $z_i = f_i(0)$ to x.
   – s_{ji}, s'_{ji} for $j = 1..n$ the shares and randomness he received from others
   Players also get public outputs C_{ik} for $i = 1..n$, $k = 0..t$ and the set *QUAL*.

Extracting $y = g^x \bmod p$:
Each player exposes $y_i = g^{z_i} \bmod p$ to enable the computation of $y = g^x \bmod p$.

2. Each player P_i, $i \in QUAL$, broadcasts $A_i = g^{f_i(0)} = g^{z_i} \bmod p$ and $B_i = h^{f'_i(0)} \bmod p$, s.t. $C_{i0} = A_i B_i$. P_i also chooses random values r_i and r'_i and broadcasts $T_i = g^{r_i}, T'_i = h^{r'_i} \bmod p$.
3. Players execute Joint-RVSS(t, n, t) for a joint random challenge d. Player P_i sets his local share of the secret challenge to d_i. All other secret output generated by this Joint-RVSS and held by P_i is erased.
4. Each player broadcast d_i. Set $d =$ EC-Interpolate$(d_1, \ldots, d_n)$.
5. P_i broadcasts $R_i = r_i + d \cdot f_i(0)$ and $R'_i = r'_i + d \cdot f'_i(0)$
6. Player P_j checks for each P_i that $g^{R_i} = T_i \cdot A_i^d$ and $h^{R'_i} = T'_i \cdot B_i^d$. If the equation is not satisfied then P_j complains against P_i.
7. If player P_i receives more than t complaints, then P_j broadcasts s_{ij}. Set $z_i =$ EC-Interpolate$(s_{i1}, \ldots, s_{in})$ and $A_i = g^{z_i}$.
8. The public value y is set to $y = \prod_{i \in QUAL} A_i \bmod p$.
9. Player P_i erases all secret information aside from his share x_i.

Fig. 1. DKG - Distributed Key Generation, $n \geq 3t + 1$

izing polynomial chosen by P_i in the run of Joint-RVSS. We will utilize these commitments to "extract" and publish the public key $y = g^x \bmod p$.

We have that $g^x = g^{\sum_{i \in QUAL} z_i} = \prod_{i \in QUAL} g^{z_i} \bmod p$. Thus, if we could have each player "deliver" g^{z_i} in a verifiable way then we could compute y. To that end, we require P_i to "split" his commitment C_{i0} into the two components $A_i = g^{z_i}$ and $B_i = h^{f'_i(0)}$ (Step 2). To ensure that he gives the correct split, P_i proves that he knows both $Dlog_g A_i$ and $Dlog_h B_i$ with Schnorr's 3-round zero-knowledge proof of knowledge of discrete-log [Sch91]. We exploit the fact that each player is proving his statement to many verifiers by generating a single challenge for all these proofs. Joint challenge d is generated with Joint-RVSS with public reconstruction (Steps 3-4). The proof completes in Steps 5-6. For the interesting properties of this form of zero-knowledge proof see Section 2. If a player fails to prove a correct split, then his value z_i is reconstructed using polynomial interpolation with the error-correcting code procedure such as [BW],

Threshold Parameters: (t, n, t')
Input: Parameters (p, q, g) and element h generated by g
Public Output: C_{ik} for $i = 1..n$, $k = 0..t'$ (referred to as the "commitment to the polynomial"). Set *QUAL* of non-disqualified players
Secret Output of P_i: x_i and x'_i the share and the associated random value
$f_i(z), f'_i(z)$ the polynomials used to share z_i
s_{ji}, s'_{ji} for $j = 1..n$ shares received from P_j

1. Each player P_i performs a Pedersen-VSS of a random value z_i as a dealer:
   (a) P_i chooses two random polynomials $f_i(z), f'_i(z)$ over Z_q of degree t': $f_i(z) = a_{i0} + a_{i1}z + ... + a_{it'}z^{t'}$ $\quad$ $f'_i(z) = b_{i0} + b_{i1}z + ... + b_{it'}z^{t'}$ Let $z_i = a_{i0} = f_i(0)$. P_i broadcasts $C_{ik} = g^{a_{ik}}h^{b_{ik}} \bmod p$ for $k = 0, ..., t'$. P_i sends shares $s_{ij} = f_i(j), s'_{ij} = f'_i(j) \bmod q$ to each P_j for $j = 1, ..., n$.
   (b) Each P_j verifies the shares received from other players for $i = 1, ..., n$

$$g^{s_{ij}}h^{s'_{ij}} = \prod_{k=0}^{t'} (C_{ik})^{j^k} \bmod p \quad (1)$$

   If the check fails for an index i, P_j broadcasts a *complaint* against P_i.
   (c) Each player P_i who, as a dealer, received a complaint from player P_j broadcasts the values s_{ij}, s'_{ij} that satisfy Eq. (1).
   (d) Each player builds the set of players *QUAL* which excludes any player
   – who received more than t complaints in Step 1b, or
   – answered to a complaint in Step 1c with values that violate Eq.(1).
2. The shared random value x is not computed by any party, but it equals $x = \sum_{i \in QUAL} z_i \bmod q$. Each P_i sets his share of the secret to $x_i = \sum_{j \in QUAL} s_{ji} \bmod q$ and the associated random value $x'_i = \sum_{j \in QUAL} s'_{ji} \bmod q$.

Fig. 2. Joint-RVSS - Joint Pedersen VSS, $n \geq 2t + 1$

which we denote by EC-Interpolate (Steps 4, 7). The players then *erase* all the information generated during the protocol except of their share of the secret key.

Proving Adaptive Security of Key Generation. In Figure 3 we present a simulator SIM-DKG for the DKG protocol.[6] This simulation is the crux of the proof of secrecy in the protocol, namely, that nothing is revealed by the protocol beyond the value $y = g^x \bmod p$ (to show this we provide the value of y as input to the simulator and require it to simulate a run of the DKG protocol that ends with y as its public output). We denote by $\mathcal{G}$ (resp. $\mathcal{B}$) the set of *currently* good (resp. bad) players. The simulator executes the protocol for all the players in $\mathcal{G}$ except one. The state of the special player P (selected at random) is used by the simulator to "fix" the output of the simulation to y, the required public key. Since the simulator does not know $Dlog_g y$ it does not know some secret

[6] If DKG is preceded with generation of h (see above), the simulator plays the part of the honest players in that protocol before running SIM-DKG to simulate DKG.

information relative to this special player (in particular the component z_P that this player contributes to the secret key). This lack of knowledge does not disable the simulator from proving that it knows P's contribution (Steps 2-6 in DKG) since the simulator can utilize a simulation of this ZK proof. However, if the adversary corrupts P during the simulation (which happens with probability $< 1/2$) the simulator will not be able to provide the internal state of this player. Thus, the simulator will need to rewind the adversary and select another special player P'. The simulation will conclude in expected polynomial time.

Input: public key y and parameters (p, q, g) and h an element generated by g

1. Perform Step 1 of DKG on behalf of the players in $\mathcal{G}$. At the end of this step the set $QUAL$ is defined. SIM-DKG knows all polynomials $f_i(z), f'_i(z)$ for $i \in QUAL$ (as it controls a majority of the players). In particular, SIM-DKG knows the values $f_i(0), f'_i(0)$.

Perform the following pre-computations:
– Choose at random one uncorrupted player $P \in \mathcal{G}$
– Compute $A_i = g^{f_i(0)}, B_i = h^{f'_i(0)}$ for $i \in QUAL \setminus \{P\}$
– Set $A^*_P = y \cdot \prod_{i \in (QUAL \setminus \{P\})} (A_i)^{-1} \bmod p$, and $B^*_P = C_{P0}/A^*_P \bmod p$
– Pick values $d, R_P, R'_P \in_R Z_q$, set $T^*_P = g^{R_P} \cdot (A^*_P)^{-d}$ and $T^{'*}_P = h^{R'_P} \cdot (B^*_P)^{-d}$

2. For each player $i \in \mathcal{G} \setminus \{\mathcal{P}\}$ execute Step 2 according to the protocol. For player P broadcast $A^*_P, B^*_P, T^*_P, T^{'*}_P$ which were computed previously.
3. Perform the Joint-RVSS(t, n, t) protocol on behalf of the uncorrupted players. Note that SIM-DKG knows the shares d_i, $i \in \mathcal{B}$. *Erase* all the secret output of the uncorrupted players in this protocol. Pick a t-degree polynomial $f^*_d(z)$ s.t. $f^*_d(0) = d$ and $f^*_d(i) = d_i$ for $i \in \mathcal{B}$. Set $d^*_i = f^*_d(i)$ for $i \in \mathcal{G}$.
4. Broadcast d^*_i for each $i \in \mathcal{G}$.
5. Broadcast $R_i = r_i + d \cdot f_i(0) \bmod q$ and $R'_i = r'_i + d \cdot f'_i(0) \bmod q$ for $i \in \mathcal{G} \setminus \{P\}$ and $R^*_P, R^{'*}_P$ for player P.
6. Execute Step 6 of DKG for all players. (Notice that the corrupted players can publish a valid complaint only against one another.)
7. For each player with more than t complaints participate in the reconstruction of their value. Note that only players in $\mathcal{B}$ can have more than t complaints.
8. Erase all information aside from the value x_i.

Fig. 3. SIM-DKG - Simulator for the Distributed Key Generation Protocol DKG

Lemma 1. *Simulator* SIM-DKG *on input* (p, q, g, h, y) *ends in expected polynomial time and computes a view for the adversary that is indistinguishable from a view of the protocol* DKG *on input* (p, q, g, h) *and output* y.

Proof: First we show that SIM-DKG outputs a probability distribution which is *identical* to the distribution the adversary sees in an execution of DKG that

produces y as an output. In the following denote by G_g the subgroup of Z_p^* generated by g.

1. The first step is carried out according to the protocol, thus values $f_i(j)$, $f'_i(j)$, $i \in \mathcal{G}$, $j \in \mathcal{B}$, and C_{ik}, $i \in \mathcal{G}$, $k = 0 \ldots t$ have the required distribution.
2. The values A_i for $i \in \mathcal{G} \setminus \{P\}$ are distributed exactly as in the real protocol. The value $A_P^* = y \cdot \prod_{i \in (QUAL \setminus \{P\})} (A_i)^{-1} = y \cdot \prod_{i \in (\mathcal{G} \setminus \{P\})} (A_i)^{-1} \cdot \prod_{i \in (\mathcal{B} \cap QUAL)} (A_i)^{-1}$ is distributed uniformly in G_g and independently from the values A_i for all i. This is because y is random and uniformly distributed and the A_i for $i \in \mathcal{B}$ are generated independently from the other ones (because Joint-RVSS is information theoretically private).
   A similar argument holds for values B_i, $i \in \mathcal{G} \setminus \{P\}$ and B_P^*.
   Finally, values T_i, T'_i, $i \in \mathcal{G} \setminus \{P\}$ are picked at random in G_g as in the protocol. Values $T_P^*, T_P^{'*}$ are also uniformly distributed in G_g. Thus the view in this step is identical.
3. Here SIM-DKG performs a Joint-RVSS(t, n, t) protocol on behalf of the players in $\mathcal{G}$ exactly as in the protocol. Thus the view in this step is identical.
4. In this step SIM-DKG broadcasts shares d_i^* of a new polynomial $f_d^*(z)$ which is random subject to the constraint that $f_d^*(j) = f_d(j)$ for $j \in \mathcal{B}$ and $f_d^*(0) = d$. Although these d_i^*'s are not the same values held as shares by the players in the previous step, the view for the adversary is still the same as in the real protocol. This is because the adversary has seen a Joint-RVSS of value d' and at most t points of the sharing polynomial f. It is easily seen that for any other value d there is another polynomial that passes through the t points held by the adversary and the free term d. Notice that since only the d_i^*'s are broadcasted the simulator does not have to "match" the public commitments generated by the Joint-RVSS.
5. Values R_i, R'_i, $i \in \mathcal{G}$ satisfy the required constraints (i.e. verification equation) as in the protocol.

We have shown that the *public* view of the adversary during the simulation is identical to the one he would see during a real execution. Now we must proceed to show that the simulator can produce a consistent view of the internal states for the players corrupted by the adversary $\mathcal{A}$. Clearly, if a player is corrupted before Step 2, the simulator can produce a consistent view because it is following the protocol. After Step 2 the simulator can show correct internal states for all the players in $\mathcal{G}$ except for the special player P. Thus, if P is corrupted the simulator rewinds the adversary to the beginning of Step 2 and selects at random a different special player. Notice that if a player P_i in $\mathcal{G} \setminus \{P\}$ is corrupted after Step 4, the simulator has broadcasted for P_i a "fake" value d_i^*. But since we erased all the private information (except the shares) generated in Joint-RVSS in Step 3, the simulator can simply claim that d_i^* was really the share held by P_i. This will not contradict any of the generated public information. □

Adaptive vs. Non-Adaptive Solutions. As noted before, DKG is based on the recent distributed key generation protocol of [GJKR99], which is secure only against a static adversary. The generation of x is the same in both protocols but

they differ in the method for extracting the public key. In the current protocol each player reveals only values $(A_i, B_i) = (g^{f_i(0)}, h^{f'_i(0)})$ from the execution of Joint-RVSS, and uses zero-knowledge proofs to guarantee that these values are properly formed. Due to this limited revealing of information it is sufficient for the simulator to "cheat" with respect to the internal state of only a single player, and yet to "hit" the desired value. However, in the protocol of [GJKR99], each player reveals A_i by publishing all values $g^{a_{ik}}$, $k = 0, ..., t$. For one of the players the simulator has to commit in this way to a polynomial without knowing $a_{i0} = f_i(0)$. Therefore he can do it in a way that is consistent with only t points on this polynomial. Thus, the simulator has an inconsistent internal state for $n-t$ players, and hence has to stop and rewind every time one of them is corrupted, which happens with overwhelming probability if the adversary is adaptive.

Key Generation with Optimal Resilience. To achieve an optimally-resilient (i.e. $n = 2t+1$) DKG protocol two changes are required. First we need to change the generation of h which occurs before DKG starts. Instead of using a VSS protocol which has a $t < n/3$ resilience ([BGW88]) we need a VSS with an optimal $t < n/2$ resilience (e.g. [CDD+99]). The second change is the following: In our DKG we publicly reconstruct a value created with Joint-RVSS protocol (see Steps 3-4). This reconstruction is currently done using error-correcting codes, which make the protocol easy to simulate, but which tolerate only $t < n/3$ faults. However, we can achieve optimal resilience by sieving out bad shares with Pedersen verification equation (Eq. (1)) if the players submit the associated random values generated by Joint-RVSS together with their shares. Therefore the players must no longer erase these values as in the current Step 3.

This change must be reflected in the simulator, because the current SIM-DKG is unable to produce these values. However, the simulator could produce them if he knew the discrete logarithm $Dlog_g(h)$. Therefore, in the h-generation protocol, instead of playing the part of the honest parties, the simulator must pick $\lambda \in Z_q$ at random, compute $h = g^\lambda$, simulate the VSS protocol of [CDD+99] to arrive at $r = h^\beta$ where $\beta = ((p-1)/q)^{-1} \bmod q$, and pass λ to the modified SIM-DKG.

4 Adaptively-Secure Threshold DSS Signature Scheme

As described in Section 2, a Threshold Signature Scheme consists of a distributed key generation, distributed signature protocol, and a signature verification procedure (see full definitions in [GJKR96] or [CGJ+]). Here we present a distributed DSS signature protocol Sig-Gen with $t < n/4$ resilience. (Below we give a brief description for how to modify this protocol to achieve optimal resilience.) We prove the unforgeability of the Threshold DSS Signature Scheme, which is combined from DKG (Section 3), Sig-Gen, and the regular DSS verification procedure DSS-Ver. The proof of robustness is deferred to [CGJ+]. We refer the reader to Section 2 for a higher-level description of the basic elements in our approach, solutions, and proofs.

Distributed Signature Protocol. The basis for the signature protocol Sig-Gen (Fig.4) is the protocol of [GJKR96], with modifications to allow for the adaptive

adversary. Protocol Sig-Gen assumes that the signing parties have previously executed the DKG protocol and hold the shares x_i of a secret key that corresponds to the generated public key y. The protocol Sig-Gen is invoked every time some message m needs to be signed.

Input: message m to be signed (plus the outputs of DKG).
Public Output: (r, s) the signature of the message m

1. **Generate** $r = g^{k^{-1}} \bmod p \bmod q$
   (a) Generate k. Players execute Joint-RVSS(t, n, t). Player P_i sets k_i to his share of the secret. All other secret information generated by the above execution is erased.
   (b) Generate random sharings of 0 on polynomials of degree $2t$. Players execute two instances of Joint-ZVSS$(t, n, 2t)$. Player P_i sets b_i and c_i to his shares of the two sharings. All other secret information generated by the above executions is erased.
   (c) Generate a random value a and $g^a \bmod p$ using DKG. Player P_i sets a_i to his share of the secret a.
   (d) Player P_i broadcasts $v_i = k_i a_i + b_i \bmod q$.
   (e) Each player computes: $\mu \stackrel{\triangle}{=} \text{EC-Interpolate}(v_1, \ldots, v_n) \bmod q$ $[= ka \bmod q]$, then $\mu^{-1} \bmod q$, and $r \stackrel{\triangle}{=} (g^a)^{\mu^{-1}}$ $[= g^{k^{-1}}] \bmod p \bmod q$.
2. **Generate** $s = k(m + xr) \bmod q$
   P_i broadcasts $s_i = k_i(m + x_i r) + c_i \bmod q$. Set $s \stackrel{\triangle}{=} \text{EC-Interpolate}(s_1, \ldots, s_n)$.
3. Player P_i erases all secret information generated in this protocol.

Fig. 4. Sig-Gen - Distributed Signature Generation, $n \geq 4t + 1$

The first part of the signature computation is the generation of the random value $r = g^{k^{-1}} \bmod p \bmod q$. This computation is very similar to the distributed key generation protocol, aside from the complication that it requires g to be raised to the inverse of the shared secret value k. We achieve this with a variation of the distributed inversion protocol from [BB89]: The players select a random uniformly distributed $k \in Z_q$ in shared form via the Joint-RVSS protocol (Step 1a). Then they perform a DKG protocol to select a random $a \in Z_q$ in shared form and publish $g^a \bmod p$ (Step 1c). The inversion of k in the exponent occurs when the players reconstruct in the clear the product $\mu = ka \bmod q$ which is a random number, invert it, and then compute $g^{k^{-1}} = (g^a)^{\mu^{-1}}$ (Steps 1d-1e). The value s is publicly reconstructed when each player reveals the product $k_i(m + x_i r)$ which lies on a $2t$-degree polynomial whose free term is $s = k(m + xr)$. As in DKG, it is crucial for the proof of adaptive security that the players erase at the end all secret information generated by this protocol.

Sig-Gen uses a Joint-ZVSS subprotocol to generate randomizing polynomials (Step 1b). This randomization is needed to hide all partial information during the public reconstruction of values μ and s (Steps 1e and 2). Joint-ZVSS, which stands for "Joint Zero VSS", is a modification of Joint-RVSS where all players

fix their values $z_i = a_{i0}$ and b_{i0} (Step 1a, Figure 2) to zero. This can be verified by other players by checking that $C_{i0} = 1 \bmod p$.

Proving Adaptive Security of Threshold DSS. We argue the following:

Theorem 1. *If DSS is unforgeable under adaptive chosen message attack then* DSS-ts=(DKG,Sig-Gen,DSS-Ver) *is a secure (unforgeable and robust)* (t, n) *- threshold signature scheme for* $t < n/4$.

Due to space limitations we present only the unforgeability part of this argument (Lemma 3) and we use the definition of unforgeability of threshold signature schemes from [GJKR96]. For the complete treatment of security of our scheme, together with more formal definitions of security of threshold signatures and a proof of robustness, we invite the reader to [CGJ+]. To prove unforgeability, we first need the following lemma about the simulator SIM-Sig presented in Fig.5:

Input: message m, its signature (r, s) of a DSS system (y, p, q, g), an element h generated by g

Compute $r^* = g^{ms^{-1}} y^{rs^{-1}} \bmod p$. Pick $\mu \in_R Z_q$, and set $\beta = (r^*)^\mu \bmod p$.

1. (a) Execute Joint-RVSS(t, n, t) on behalf of the uncorrupted players. This results in shares k_i for each of the uncorrupted players. Erase all secret values aside from k_i.
   (b) Execute two sharings of a 0 value on polynomials of degree $2t$ using Joint-ZVSS$(t, n, 2t)$. This results in shares b_i, c_i for the uncorrupted players. Erase all secret values aside from b_i, c_i.
   (c) Run the simulator SIM-DKG of the DKG protocol on input β as the "public key". This results in shares a_i for each of the uncorrupted players. Note that all other information has already been erased by the simulator which was called as a sub-routine.
   (d) The simulator knows values $v_i = k_i a_i + b_i$, $i \in \mathcal{B}$ that should be broadcast by the players controlled by the adversary in Step 1e of the signing protocol. Choose a $2t$-degree polynomial $f_v(z)$ s.t. $f_v(0) = \mu$, $f_v(i) = v_i$ for $i \in \mathcal{B}$. Set $v_i^* = f_v(i)$ for $i \in \mathcal{G}$. Compute $b_i^* = v_i^* - k_i a_i$, $i \in \mathcal{G}$. Erase the secret values b_i for $i \in \mathcal{G}$. Broadcast v_i^* for $i \in \mathcal{G}$.
2. The simulator knows values $s_i = k_i(m + x_i r) + c_i$, $i \in \mathcal{B}$ that should be broadcast by the players controlled by the adversary in Step 2 of the signing protocol. Choose a $2t$-degree polynomial $f_s(z)$ s.t. $f_s(0) = s$, $f_s(i) = s_i$ for $i \in \mathcal{B}$. Set $s_i^* = f_s(i)$ for $i \in \mathcal{G}$. Compute $c_i^* = s_i^* - k_i(m + x_i r)$, $i \in \mathcal{G}$. Erase the secret values c_i for $i \in \mathcal{G}$. Broadcast s_i^* for $i \in \mathcal{G}$.
3. Erase all the information generated by the signature generation.

Fig. 5. SIM-Sig - Simulator for the Distributed Signature Protocol Sig-Gen

Lemma 2. *Simulator* SIM-Sig *on input* $(p, q, g, h, y, m, (r, s))$ *ends in expected polynomial time and computes a view for the adversary that is indistinguishable from a view of the protocol* Sig-Gen *on input public key* (p, q, g, h, y)*, message* m*, and output signature* (r, s).

Proof: 1. Generation of r

(a) The simulator executed Joint-RVSS according to the protocol, thus all the values generated by these executions have the required distribution.
(b) The simulator executed two instances of Joint-ZVSS according to the protocol, thus all the values generated by these executions have the required distribution.
(c) It has been proved that the simulator SIM-DKG outputs the desired distributions for the DKG protocol. Notice also that β is uniformly distributed in G_g (the same as g^a in the real protocol).
(d) Here the simulator broadcasts values v_i^* which were not computed according to the protocol. Yet because these shares have been chosen at random under the condition that they interpolate to a random free term μ and the polynomial interpolated by them matches the shares held by the adversary, the view of the adversary is exactly the same as in the real protocol.

2. Generation of s: The same argument as above applies to the values s_i^*.

The discussion about the internal states presented by the simulator to the adversary when a player is corrupted is identical to the one in the proof of Lemma 1. It is important to notice that the only rewinding happens during the simulation of the DKG subroutine inside this protocol. It should be restated here that once the DKG simulator completes its execution the adversary can now corrupt any of the players, including the "special" player, because now even for that player the simulator has a consistent view. □

Lemma 3. *If DSS is unforgeable under adaptive chosen message attack then* DSS-ts=(DKG, Sig-Gen, DSS-Ver) *is an* unforgeable (t, n)*-threshold signature scheme for* $t < n/4$.

Proof: Assume that DSS-ts is not unforgeable. Then there exists a t-threshold adversary $\mathcal{A}$ s.t. with a non-negligible probability $\mathcal{A}$ outputs a valid DSS (message,signature) pair $(m, (r, s))$, after participating in the initial execution of DKG and then in the repeated execution of Sig-Gen on messages $m_1, m_2, ...$ of $\mathcal{A}$'s choice. Furthermore, none of the m_i's is equal to m. Using such adversary $\mathcal{A}$, we show how to construct a forger $\mathcal{F}$ against the *regular* DSS scheme. $\mathcal{F}$ is given as input a DSS system (p, q, g), and a random public key y. Additionally, $\mathcal{F}$ can accesses a signature oracle O_{Sig} that provides DSS signatures under the given public key (p, q, g, y).

$\mathcal{F}$ fixes the random coins of $\mathcal{A}$. First $\mathcal{F}$ plays the part of the honest parties in the h-generation protocol. Then $\mathcal{F}$ runs an interaction between SIM-DKG and $\mathcal{A}$ on input (p, q, g, h, y). By lemma 1, the simulation ends in expected polynomial time and $\mathcal{A}$ receives a view that is identical to $\mathcal{A}$'s view of a random execution of DKG that outputs y. When $\mathcal{A}$ requests a signature on message m_i, $\mathcal{F}$ submits m_i to O_{Sig} and receives (r_i, s_i), a random DSS signature on m_i. Then $\mathcal{F}$ then runs an interaction between SIM-Sig and $\mathcal{A}$ on input $(p, q, g, h, y, m_i, (r_i, s_i))$. By lemma 2, this simulation ends in expected polynomial time and $\mathcal{A}$ receives a view that is identical to $\mathcal{A}$'s view of a random execution of Sig-Gen that outputs (r_i, s_i). Finally, since its views of this simulation are indistinguishable from the

real ones, $\mathcal{A}$ outputs a valid DSS signature $(m,(r,s))$ where $m \neq m_i$ for all i. $\mathcal{F}$ outputs this $(m,(r,s))$ which is a successful existential forgery since $\mathcal{F}$ never asked its oracle this message m. □

Threshold DSS with Optimal $t < n/2$ Resilience. Because of lack of space we defer the description of the modifications that achieve optimal resilience in the Sig-Gen protocol to the full version [CGJ+] of this paper. These modifications include the use of the $t < n/2$ resilient version of DKG as described at the end of section 3. Furthermore, we use the results of [GRR98] to increase the resilience of the secret-multiplication steps of the signature protocol itself.

5 Further Applications

The techniques introduced in this paper enable us to achieve adaptive security for other threshold public-key systems. Here we sketch our solution to Adaptive Threshold RSA and Adaptive Proactive Solutions.

Adaptive Threshold RSA. We can achieve an adaptively-secure Threshold RSA Signature Generation protocol (but without distributed key generation) with optimal resilience. Furthermore, our protocol runs in a constant number of rounds. We build our solution on the Threshold RSA solution of [Rab98]. The protocol of that paper needs to be modified to use a Pedersen VSS wherever a Feldman VSS is used, the zero-knowledge proofs which appear need to be modified accordingly, and commitments should be of the Pedersen form.

The most interesting change required in this protocol is due to the following: If the current protocol is invoked twice on two different messages, each player gives its partial signature on these messages under its fixed partial key. But as we have seen in Section 2, the simulator is not allowed to rewind back beyond the current invocation of the signature protocol that it is simulating. Clearly, the simulator cannot know the partial keys of all players. If a player for whom he does not know the partial key is broken into during the simulation of the signing of the second message, then the simulator would need to explain both partial signatures of that player, and hence would be forced to rewind beyond the current invocation of the signature protocol.

To avoid this problem, the partial keys need to be changed after each signature generation. This is achieved in a straightforward manner (though it adds a performance penalty to the protocol).

Adaptive Proactive Solutions. Our threshold DSS scheme, as well as other discrete–log based threshold schemes built with our techniques, can be easily proactivized [HJJ+97] by periodic refreshment of the shared secret key. Due to space limitations we simply state here that such refreshment can be achieved in the adaptive model if the players execute a Joint-ZVSS protocol (see Section 4), and add the generated shares to their current share of the private key x.

References

BB89. J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds. In *Proc. 8th ACM PODC*, pages 201–209. ACM, 1989.

BGW88. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Noncryptographic Fault-Tolerant Distributed Computations. In *Proc. 20th STOC*, pages 1–10. ACM, 1988.

BH92. D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Eurocrypt '92*, pages 307–323, 1992. LNCS No. 658.

BW. E. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent 4,633,470.

Can98. R. Canetti. Security and composition of multiparty cryptographic protocols. Available at the Theory of Cryptography Library, http://theory.lcs.mit.edu/~tcryptol. 1998.

CCD88. D. Chaum, C. Crepeau, and I. Damgard. Multiparty Unconditionally Secure Protocols. In *Proc. 20th STOC*, pages 11–19. ACM, 1988.

CD98. R. Cramer and I. Damgard. Zero-knowledge proof for finite fields arithmetic, or: Can zero-knowledge be for free. In *Crypto '98*, pages 424–441, 1998. LNCS No. 1462.

CDD+99. R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations with dishonest minority. In *Eurocrypt '99*, pages 311–326, 1999. LNCS No.

CFGN96. Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proc. 28th STOC*, pages 639–648. ACM, 1996.

CGJ+. R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. http://www.research.ibm.com/security/adss.ps.

DNS98. C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *Proc. 30th STOC*, pages 409–418. ACM, 1998.

FMY. Y. Frankel, P. MacKenzie, and M. Yung. Adaptively-secure distributed public-key systems. Personal communication with M.Yung.

GJKR96. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Eurocrypt '96*, pages 354–371, 1996. Full version: http://www.research.ibm.com/security/DSSthresh.ps.

GJKR99. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. The (in)security of distributed key generation in dlog-based cryptosystems. In *Eurocrypt '99*, pages 295–310, 1999. LNCS No.

GK96. O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM. J. Computing*, 25(1), 1996.

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

GMR89. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM. J. Computing*, 18(1):186–208, February 1989.

GRR98. R. Gennaro, M. Rabin, and T Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proc. 17th ACM PODC*, pages 101–112. ACM, 1998.

HJJ+97. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *1997 ACM Conference on Computers and Communication Security*, 1997.

IS93. T. Itoh and K. Sakurai. On the complexity of constant round zkip of possession of knowledge. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E76-A(1), January 1993.

KPR98. J. Kilian, E. Petrank, and C. Rackoff. Lower bounds for zero-knowledge on the internet. In *Proc. 39th FOCS*, pages 484–492. IEEE, 1998.

Ped91a. T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Crypto '91*, pages 129–140, 1991. LNCS No. 576.

Ped91b. T. Pedersen. A threshold cryptosystem without a trusted party. In *Eurocrypt '91*, pages 522–526, 1991. LNCS No. 547.

Rab98. Tal Rabin. A simplified approach to threshold and proactive RSA. In *Crypto '98*, pages 89–104, 1998. LNCS No. 1462.

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