Evaluating Security of Voting Schemes in the Universal Composability Framework

Jens Groth[*]

BRICS[†], University of Aarhus and Cryptomathic A/S[‡]

Remark. Appears in ACNS 2004, this is the full paper. It contains a proof of Theorem 1, some more details on security against adaptive adversaries in the erasure-free model, and a conclusion.

Abstract

In the literature, voting protocols are considered secure if they satisfy requirements such as privacy, accuracy, robustness, etc. It can be time consuming to evaluate a voting protocol with respect to all these requirements and it is not clear that the list of known requirements is complete. Perhaps because of this many papers on electronic voting do not offer any security proof at all.

As a solution to this, we suggest evaluating voting schemes in the universal composability framework. We investigate the popular class of voting schemes based on homomorphic threshold encryption. It turns out that schemes in this class realize an ideal voting functionality that takes the votes as input and outputs the result. This ideal functionality corresponds closely to the well-known ballot box model used today in manual voting. Security properties such as privacy, accuracy and robustness now follow as easy corollaries. We note that some security requirements, for instance incoercibility, are not addressed by our solution.

Security holds in the random oracle model against a non-adaptive adversary. We show with a concrete example that the schemes are not secure against adaptive adversaries. We proceed to sketch how to make them secure against adaptive adversaries in the erasure model with virtually no loss of efficiency. We also sketch how to achieve security against adaptive adversaries in the erasure-free model.

Keywords: Voting, homomorphic threshold encryption, universal composability.

---

[*] Jens Groth, homepage: `www.brics.dk/~jg`, e-mail: `jg@brics.dk`.

[†] Basic Research in Computer Science (`www.brics.dk`), funded by the Danish National Research Foundation.

[‡] Cryptomathic A/S, Jægergårdsgade 118, 8000 Århus C, Denmark (`www.cryptomathic.com`)

1 Introduction

We consider the security of voting protocols. As time has progressed, more and more security requirements have been published in the literature. Examples of such requirements are privacy, accuracy, fairness, robustness, universal verifiability, incoercibility and receipt-freeness [BM03, LGT$^+$03]. With this growing list of requirements, designers of voting protocols face two problems: if they do not know the literature well they may miss a security requirement, and even if they do cover all known requirements this does not guarantee that new yet to be discovered requirements are satisfied by their voting scheme.

To partially solve these problems we suggest evaluating voting schemes in the universal composability (UC) framework of Canetti [Can01]. In the UC framework, an execution of a multi-party computation protocol is compared to an execution where a trusted ideal functionality handles the data and produces the output. A protocol is said to be secure if an adversary operating in a real-life model can be simulated in the ideal process model with the ideal functionality. In the case of voting, the ideal functionality takes as input the votes and outputs the result of the election. This ideal functionality corresponds to the old method of voters marking their choice on paper and putting the ballot in a box, which is opened once the election is over.

Let us see how this solution addresses some of the properties that we mentioned. Privacy and accuracy are automatically satisfied since it is a part of the model that input to the ideal functionality is not revealed in any way to the adversary and the ideal functionality does compute the result correctly. Robustness follows too; in the UC framework, we can corrupt parties and still have a good simulation in the ideal process. Fairness follows from the fact that the ideal functionality does not reveal any partial tallies during the process.

Our approach has the advantage that it covers many security requirements in a single security model. This simplifies security proofs since we only need to prove universal composability to prove all these specific security requirements. Our approach is also pro-active in the sense that using a general security model may mean that security requirements yet to be discovered are covered.

We do not claim to solve all security issues with this approach. In particular, universal composability of a voting scheme does not guarantee universal verifiability, incoercibility, receipt-freeness or protection against hackers. However, considering that many security issues are dealt with, and considering that the properties dealt with are often defined vaguely in papers dealing with voting schemes, we do find that this application of the UC framework is worthwhile to investigate.

The UC framework allows for modular composition. In short, this means that if we take a hybrid protocol, where part of the protocol is specified by an ideal functionality, then we can freely plug in any protocol that securely realizes this ideal functionality. Most voting schemes presented in the literature make shortcuts. They assume we have a broadcast channel with memory or an anonymous broadcast channel. Often they also assume some public keys are set up and assume that voters are registered without specifying how this is done. We take this approach too and assume these things are provided through an ideal functionality. The modular composition theorem of the UC framework tells us that this is a sound approach and that we may later insert any protocol that realizes this functionality to get a full-blown election protocol.

The specific class of voting protocols we look at in this paper is based on homomorphic threshold encryption. Many such schemes have been proposed in the literature [CGS97, DJ01, BFP+01, DGS03], only the first one of these offers a security proof. We prove that indeed these schemes realize an ideal voting functionality when the adversary is non-adaptive. The schemes are not secure against adaptive adversaries, however, we propose a simple modification to make them secure against adaptive adversaries in the erasure model. We suggest another modification based on Paillier encryption that gives security against adaptive adversaries in the erasure-free model.

2 Preliminaries

In this section, we present the various tools used in the class of voting schemes we intend to investigate. Before doing so, we offer a brief introduction to the idea behind this class of voting protocols.

The idea behind voting based on homomorphic encryption. We assume that the parties have access to a message board where everybody may post messages, everybody can read the messages posted on the message board, messages cannot be deleted, and all messages are authenticated, for instance with digital signatures. All communication will take place through this message board. Public data pertaining to the election is also posted on the message board. In particular, a public key *pk* for a cryptosystem is posted.

In this example, we assume for simplicity that the voters only have two choices. We encode "yes" as 1, while "no" is encoded as 0. A voter casts his vote by encrypting the vote and posting it on the message board, i.e., posting $E_{pk}(0)$ or $E_{pk}(1)$. Since the messages are authenticated, everybody can check whether an eligible voter cast the vote.

The cryptosystem should have a homomorphic property:

$$E_{pk}(m_1; r_1) \cdot E_{pk}(m_2; r_2) = E_{pk}(m_1 + m_2; r_1 + r_2).$$

When everybody has cast his vote we may therefore compute the product of all the ciphertexts and get an encryption of the number of "yes" votes.

Now the authorities must decrypt this ciphertext containing the result of the election. For this purpose, we assume that the cryptosystem has threshold decryption. The authorities each hold a secret share of the private key and if sufficiently many of them cooperate, they may decrypt the ciphertext. However, no coalition below the threshold value is able to decrypt any of the encrypted votes; this preserves privacy.

To prevent cheating we require that voters attach a non-interactive zero-knowledge proof that their ciphertext contains either 0 or 1. Otherwise, it would for instance be easy to cast 100 "yes"-votes by posting $E_{pk}(100)$. Standard non-interactive zero-knowledge proofs are too cumbersome to be used in practice, therefore this is typically done through a 3-move honest verifier zero-knowledge proof of correctness of a vote made non-interactive through the Fiat-Shamir heuristic.

In this section, we define Σ-protocols [CDS94], the type of 3-move honest verifier zero-knowledge proofs that we use. We then note that these proofs in the random oracle model [BR93] can be transformed into non-interactive zero-knowledge proofs. We prove that in the random oracle model, we are dealing with a proof of knowledge, and for any prover there exists an emulator that also produces corresponding witnesses. This can be seen as a random oracle parallel of witness extended emulation as defined by Lindell [Lin01]. Finally, we define the kind of homomorphic threshold encryption that we need.

Σ-protocols. A Σ-protocol is a special type of 3-move proof system. Say we have an element x and a language L. The prover P knows a witness w for $x \in L$ and wants to convince the verifier V that $x \in L$. We assume that both parties have access to a common reference string σ chosen with a suitable distribution. Some Σ-protocols do not require this, and in that case, we can of course just let σ be the empty string. The protocol goes like this: The prover sends an initial message a, receives a random challenge e and produces an answer z. V can now evaluate (σ, x, a, e, z) and decide whether to accept or reject the proof.

A Σ-protocol satisfies the following properties.

Completeness: Given (x, w) where w is a witness for $x \in L$ the prover will with overwhelming probability convince the verifier, if they both follow the protocol.

Special Soundness: There exists an efficient extractor that for any x given two acceptable proofs (a, e, z) and (a, e', z') with the same initial message but different challenges can compute a witness w for $x \in L$.

Special Honest Verifier Zero-Knowledge: There exists an efficient simulator that given x, e can create a "proof" (a, e, z) for $x \in L$, which is indistinguishable from a real proof with challenge e.

Non-interactive zero-knowledge proofs. Given access to a random oracle $\mathcal{O}$ we can transform a Σ-protocol into a non-interactive proof system. To get the challenge e we form the initial message a, query $\mathcal{O}$ with (x, a, aux) to get the challenge e and then compute the answer z.[1] The proof is then (a, z, aux). To verify such a proof query $\mathcal{O}$ with (x, a, aux) to get e and then run the verifier from the Σ-protocol.

Using standard techniques, we can prove that we get a non-interactive proof system with the following properties:

Completeness: Given (x, w) where w is a witness for $x \in L$ the verifier will accept if both the prover and the verifier follow the protocol.

Soundness: A dishonest prover cannot convince the verifier if $x \notin L$.

Zero-Knowledge: There exist a simulator $S^{\mathcal{O}}$ that given $x \in L$ can create a convincing proof (a, z, aux) indistinguishable from a real proof provided it has the following ability

---

[1] Typically, aux will contain the identity of the prover in order to prevent somebody else to duplicate the proof and claim to have made it.

to modify the oracle. It may give (x, a, aux, e) to $\mathcal{O}$ and provided (x, a, aux) has not been queried before $\mathcal{O}$ assigns the value e to be the answer to query (x, a, aux).

The random oracle model is an idealization of the Fiat-Shamir heuristic, see [BR93]. In the Fiat-Shamir heuristic the prover uses a cryptographic hash-function to produce the challenge as $e = \mathrm{hash}(x, a, aux)$.

Witness extended emulation in the random oracle model. A Σ-protocol is a proof of knowledge in the random oracle model. We formulate this in the form of witness extended emulation in the following way. Given some adversary that produces a vector of elements $x \in L$ and valid proofs of memberships of L, there is an emulator E_A that produces identically distributed elements together with the corresponding witnesses for memberships of L.

Theorem 1 *For all adversaries A there exists an expected polynomial time emulator E_A such that for all distinguishers D (even unbounded ones) we have*

$$\begin{aligned} & P[(\vec{x}, \vec{p}, s) \leftarrow A^{\mathcal{O}}(z) : (\vec{x}, \vec{p}) \in V \wedge D^{\mathcal{O}}(\vec{x}, \vec{p}, s, z) = 1] \\ \approx\ & P[(\vec{x}, \vec{p}, \vec{w}, s) \leftarrow E_A^{\mathcal{O}}(z) : (\vec{x}, \vec{p}) \in V \wedge (\vec{x}, \vec{w}) \in W \wedge D^{\mathcal{O}}(\vec{x}, \vec{p}, s, z) = 1], \end{aligned}$$

where z is some advice with length bounded by a polynomial in k, $\mathcal{O}$ is a random oracle, V is the set of vector pairs $(\vec{x}, \vec{p})$ such that $\vec{p}$ contains valid proofs for the elements in $\vec{x}$ belonging to L, and W is the set of pairs $(\vec{x}, \vec{w})$ where $\vec{w}$ contains witnesses for the elements of $\vec{x}$ belonging to L.[2]

We construct the emulator E_A as follows: It runs a copy of $A^{\mathcal{O}}(z)$ to get output $(\vec{x}, \vec{p}, s)$. During the run, it saves the state of A whenever it makes a query to $\mathcal{O}$. E_A will now attempt to find witnesses for the elements in $\vec{x}$ belonging to L. If any of the proofs have been made without querying the oracle, i.e., A simply guessed the oracle value we surrender, however, this only has negligible probability of happening.

Let us consider the case where A did make a query to get the challenge e in each of its proofs. Consider a query (x, a, aux) that A used in a proof $p = (a, z, aux)$. Since we saved the state of A after this query we can run A again this time giving random answers to all subsequent queries to $\mathcal{O}$. We can hope that in the end A includes a new proof (a, z', aux) where we have selected $e' \neq e$. In that case we may by the special soundness property extract a witness w for $x \in L$. E_A simply repeats this procedure for each proof in $\vec{p}$ until it has new proofs for all of the elements in $\vec{x}$. With overwhelming probability, these new proofs use different challenges and therefore witnesses can be extracted.

---

[2] It is instructive to consider this theorem in connection with the cryptosystem TDH0 in [SG02]. TDH0 is a cryptosystem where a Σ-protocol made non-interactive with a random oracle is used to prove knowledge of the plaintext. Intuitively one might argue CCA2 security by saying that the adversary already knows the answer when submitting decryption requests. However, Gennaro and Shoup show that this argument fails since rewinding is used to get the plaintexts, and since decryption requests may depend on oracle queries made before several other oracle queries we risk an exponential blow-up when tracking back through the decryption requests. Our theorem does not solve this problem. What our theorem can be used to prove, however, is that TDH0 is non-malleable.

We have to argue that E_A runs in expected polynomial time. Let $time_A(k)$ be a polynomial bound on the running time of A. Consider the state of A after having made a query to the random oracle. Let p be the probability that A is going to use the answer to make a valid proof. In case A does use the answer E_A uses an expected $1/p$ simulated runs of A from this point to get an oracle answer that is used to make a valid proof in a simulated run. Since each simulated run of A takes at most time $time_A(k)$, the expected time used by E_A on this state is at most $p\frac{1}{p}time_A(k)$. Since the running time of A is $time_A(k)$, it can make at most $time_A(k)$ queries to the random oracle. Therefore, $time_A(k)^2$ is an upper bound on the expected running time of E_A. □

Homomorphic threshold encryption. A (t, n)-threshold cryptosystem is a public key cryptosystem where the secret key is shared between n authorities $A_1, \ldots, A_n$. If t of them cooperate they may decrypt ciphertexts, but any group of less than t authorities cannot learn anything about the contents of a ciphertext.

We use a key generation algorithm K to generate the keys. In general, all elements of the cryptosystem, messages, randomness and ciphertexts belong to suitable groups. We write the ciphertext space with multiplicative notation and the other groups with additive notation. The key generation algorithm produces a public key pk which is used for encryption, secret keys $sk_1, \ldots, sk_n$ used for decryption, and verification keys $vk_1, \ldots, vk_n$ that are public and used for verifying that the authorities act according to the protocol.

Encryption works as usual. To decrypt a ciphertext the authorities use their secret keys to produce decryption shares. Given t decryption shares anybody can combine them to get the plaintext. The verification keys are used by the authorities to make a zero-knowledge proof that they have provided the correct decryption shares.

We require that the cryptosystem have the following properties.

Semantic security: The cryptosystem must be semantically secure.

Errorless decryption: With overwhelming probability, the key generation algorithm selects keys such that there is probability 1 for the decryption to yield the message encrypted.[3]

Homomorphicity: For all messages m_1, m_2 and randomizers r_1, r_2 we have $E_{pk}(m_1+m_2; r_1+r_2) = E_{pk}(m_1; r_1) \cdot E_{pk}(m_2; r_2)$.

Simulatability of decryption: There is an algorithm S that takes as input a ciphertext c, a message m and the secret shares of any group of $t-1$ authorities and produces simulated decryption shares for all the remaining authorities that c decrypts to m. S must be such that even with knowledge of the corresponding $t-1$ keys the simulated decryption shares are indistinguishable from real decryption shares.

---

[3] Most known cryptosystems have this property. However, in the notion of deniable encryption [CDNO97] the goal is to make it possible to deny that a particular thing was encrypted by producing honest looking randomness for an entirely different plaintext.

3 Universal Composability

The universal composability framework is described in details in [Can01]. The main gist is to compare a real-life execution of a protocol with an ideal process. We say a real-life protocol π realizes an ideal functionality $\mathcal{F}$ if an adversary $\mathcal{A}$ in the real-life model cannot gain more than an adversary $\mathcal{S}$ in the ideal process does. More precisely, we have an environment $\mathcal{Z}$ that gives inputs to parties, sees outputs from parties and learns which parties are corrupted, and we say π securely realizes $\mathcal{F}$ if $\mathcal{Z}$ cannot distinguish the real-life protocol with adversary $\mathcal{A}$ from the ideal process with simulator $\mathcal{S}$.

In the ideal process, the ideal functionality handles everything taking place in the protocol. The parties in the protocol hand their inputs from $\mathcal{Z}$ directly and securely to $\mathcal{F}$. $\mathcal{F}$ computes the parties outputs and sends it to them. When a party receives a message from $\mathcal{F}$, it outputs this message. $\mathcal{S}$ is restricted to corrupting some of the parties and blocking messages from $\mathcal{F}$ to the honest parties. On the other hand, in the real-life execution the parties carry out the protocol π to produce their outputs.

One main feature in this framework is security under modular composition. Let us say we have a protocol ρ that realizes the ideal functionality $\mathcal{F}$. Say that ρ is used as a sub protocol in π and write this as π^ρ. We may then form the hybrid $\pi^{\mathcal{F}}$ where calls to ρ are replaced with calls to $\mathcal{F}$. It is a theorem that π^ρ securely realizes $\pi^{\mathcal{F}}$.

Key generation and message board hybrid model. We will take advantage of the modular composition theorem and work in a hybrid model where we assume we have protocols that realize the key generation and message board functionality described in Figure 1. For distributed key generation protocols refer to [BF97, GJKR99, FS01, ACS02]. This enables us to concentrate on the voting protocol itself.

We note that in $\mathcal{F}_{\mathrm{KM}}$ we allow $\mathcal{A}$ to block voters' messages. This is to cover all the benign and malicious failures that may occur when voters try to cast their vote; everything from the Internet connection being unstable to an adversary deliberately cutting the cables to groups of voters with a particular political opinion. A typical requirement of a voting system is that it should be available, i.e., voters wanting to vote should have access to vote. This covers protecting against denial of service attacks, etc., but is not part of what the cryptographic protocol can accomplish. Therefore, we specifically allow the adversary to block votes. We quantify over all adversaries in the security proof, so in particular the security proof also holds for non-blocking adversaries that do not block messages, i.e., it holds for voting systems with the availability property. In contrast, for simplicity we do not allow the adversary to block inputs from the authorities. This choice is reasonable since any voting system must have appropriate back-up procedures to ensure that all authorities can communicate as needed.

Another remark pertains to resubmission of votes. Depending on the requirements, sometimes dictated by law, it may or may not be allowed for voters to change their votes. For simplicity, we treat the case where voters cannot change their mind, and therefore we only allow a single message not to be blocked. Security can be proved quite similarly in the case where we allow voters to change their mind.

Functionality $\mathcal{F}_{\mathrm{KM}}$

$\mathcal{F}_{\mathrm{KM}}$ proceeds as follows, running with parties $V_1, \ldots, V_m, A_1, \ldots, A_n$ and an adversary $\mathcal{A}$.

- Generate keys for the homomorphic threshold cryptosystem $(pk, vk_1, \ldots, vk_n, sk_1, \ldots, sk_n)$. Send (**public key**, sid, pk) to all parties and $\mathcal{A}$. Send (**verification keys**, $sid, vk_1, \ldots, vk_n$) to all the authorities and $\mathcal{A}$. For $i = 1, \ldots, n$ send (**secret share**, sid, sk_i) to A_i.
- Upon receiving (**message**, sid, m) from party V_i store (**message**, sid, V_i, m) and send it to $\mathcal{A}$.
- Upon receiving (**no-block**, sid, V_i, m) from $\mathcal{A}$ check whether (**message**, sid, V_i, m) has been stored. In that case, store (**post**, sid, V_i, m) and ignore subsequent (**no-block**, $sid, V_i, \ldots$) messages from $\mathcal{A}$.
- Upon receiving (**tally**, sid) from $\mathcal{A}$, send all stored (**post**, sid, V_i, m) messages to $A_1, \ldots, A_n$. Ignore subsequent (**tally**, sid) requests.
- Upon receiving (**post**, sid, m) from party A_i send (**post**, sid, A_i, m) to $A_1, \ldots, A_n$ and $\mathcal{A}$.

Figure 1: The key generation and message board functionality, $\mathcal{F}_{\mathrm{KM}}$.

Functionality $\mathcal{F}_{\mathrm{VOTING}}$

$\mathcal{F}_{\mathrm{VOTING}}$ proceeds as follows, running with parties $V_1, \ldots, V_m, A_1, \ldots, A_n$ and an adversary $\mathcal{S}$.

- Upon receiving (**vote**, sid, V_i, v) from V_i store it and send (**vote**, sid, V_i) to $\mathcal{S}$. Ignore future (**vote**, $sid, \ldots$) messages from V_i.
- Upon receiving (**no-block**, sid, V_i) from $\mathcal{S}$ check whether some (**vote**, sid, V_i, v) has been stored. In that case, add v to the result and ignore subsequent (**no-block**, sid, V_i) messages from $\mathcal{S}$.
- Upon receiving (**result**, sid) from $\mathcal{S}$ compute the result and send (**result**, sid, result) to $\mathcal{S}$ and $A_1, \ldots, A_n$ and halt.

Figure 2: The voting functionality, $\mathcal{F}_{\mathrm{VOTING}}$.

Voting protocol. Before describing the protocol that we use to realize the ideal voting functionality in Figure 2, we need to discuss how to encode the voters' choice as a plaintext to be encrypted. In [DJ01, BFP$^+$01, DGS03] this is done by assigning each candidate a number $j \in \{0, \ldots, L-1\}$ and encoding the candidate as M^j, where M is a strict upper bound on the number of votes any candidate can receive. Adding many such votes gives a result on the form $\sum_{j=0}^{L-1} v_j M^j$ where v_j is the number of votes on candidate number j. Votes and result can be embedded in a message space on the form $\mathbb{Z}_N$ provided $N \geq M^L$. More generally we require that there is an encoding such that:

- Each valid vote v can be encoded as $\mathrm{Encode}(v)$.
- The sum of the encodings yields an encoding of the result, $\mathrm{Encode}(\mathrm{result})$.
- It is possible to efficiently extract the result from an encoding.
- The encodings can be embedded in the message space of the cryptosystem.

We describe the voting protocol based on homomorphic threshold encryption in Figure 3. Examples of such voting protocols can be found in [CGS97, DJ01, BFP$^+$01, DGS03].

Voting Protol $\pi_{\mathrm{VOTING}}^{\mathcal{F}_{\mathrm{KM}}}$

The voting protocol for voters $V_1, \ldots, V_m$ and authorities $A_1, \ldots, A_n$ with access to ideal functionality $\mathcal{F}_{\mathrm{KM}}$ and random oracle $\mathcal{O}$ is as follows.

1. Invoke $\mathcal{F}_{\mathrm{KM}}$ to give each voter $V_1, \ldots, V_m$ the public key and give each authority $A_1, \ldots, A_n$ all the verification keys and its own secret key.
2. Each voter V_i with a public key pk on the incoming message tape and a valid vote v_i on the input tape computes $c_i \leftarrow E_{pk}(\mathrm{Encode}(v_i))$. He creates a proof p_i for the vote being correct using a Σ-protocol with $\mathcal{O}$'s answer on (c_i, a_i, pk, sid, V_i) as the challenge e_i.

   He sends $(\mathbf{message}, sid, c_i, p_i)$ to $\mathcal{F}_{\mathrm{KM}}$.
3. Authority A_j with the public key and the verification keys on its tape and a secret share of the private key on its tape does the following. When receiving a bunch of broadcast votes it computes C as the product of all the votes with valid proofs. Then it computes the decryption share ds_j. It also forms a proof p_j for the decryption share being correct using the verification key vk_j. The challenge in this proof is computed with $\mathcal{O}$.

   It sends $(\mathbf{post}, sid, ds_j, p_j)$ to $\mathcal{F}_{\mathrm{KM}}$.
4. Each authority picks the first t decryption shares with valid proofs that it receives and computes the plaintext of C.

   It interprets the plaintext as $\mathrm{Encode}(\mathrm{result})$ and outputs $(\mathbf{result}, sid, \mathrm{result})$.

Figure 3: The voting protocol $\pi_{\mathrm{VOTING}}^{\mathcal{F}_{\mathrm{KM}}}$.

Ideal process adversary. To prove security of the voting protocol we need to provide an ideal process adversary $\mathcal{S}$ that fares as well as $\mathcal{A}$ does in the $\mathcal{F}_{\mathrm{KM}}$-hybrid model. $\mathcal{S}$ is described in Figure 4.

Ideal process adversary $\mathcal{S}$

$\mathcal{S}$ operates in the ideal process with dummy voters $\tilde{V}_1, \ldots, \tilde{V}_m$ and dummy authorities $\tilde{A}_1, \ldots, \tilde{A}_n$. It has input z. It controls the random oracle $\mathcal{O}$ in the sense that it may assign a response e to a query q. This means that it can simulate proofs.
$\mathcal{S}$ runs a simulated $\mathcal{F}_{\text{KM}}$-hybrid execution with simulated adversary $\mathcal{A}$. We write $V_1, \ldots, V_m$ and $A_1, \ldots, A_n$ to denote simulated parties.

- $\mathcal{S}$ forwards all messages between $\mathcal{A}$ and $\mathcal{Z}$.
- $\mathcal{S}$ simulates the invocation of $\mathcal{F}_{\text{KM}}$. Having done this it knows the secret shares of the private key of all the authorities, in other words $\mathcal{S}$ may decrypt messages encrypted under the public key.
- Suppose $\mathcal{A}$ on behalf of a corrupt V_i sends $(\textbf{message}, sid, c_i, p_i)$ and sends $(\textbf{no-block}, sid, V_i, c_i, p_i)$ to $\mathcal{F}_{\text{KM}}$. $\mathcal{S}$ checks whether the proof is valid and in that case it decrypts c_i to get a vote v_i. It submits $(\textbf{vote}, sid, V_i, v_i)$ to $\mathcal{F}_{\text{VOTING}}$ on behalf of $\tilde{V}_i$ and sends $(\textbf{no-block}, sid, V_i)$ to $\mathcal{F}_{\text{VOTING}}$.
- Upon receiving $(\textbf{vote}, sid, V_i)$ from $\mathcal{F}_{\text{VOTING}}$ it knows that $\tilde{V}_i$ got $(\textbf{vote}, sid, V_i, v_i)$ as input from $\mathcal{Z}$. It does not know the actual vote v_i.

  As long as V_i has not received the public key for the election $\mathcal{S}$ ignores the problem, but if V_i has or gets the public key for the election $\mathcal{S}$ must simulate V_i trying to cast a vote. It forms $c_i = E_{pk}(0)$ and simulates a proof p_i for c_i containing a valid vote. It simulates V_i sending $(\textbf{message}, sid, V_i, c_i, p_i)$ to $\mathcal{F}_{\text{KM}}$ and sends the resulting $(\textbf{message}, sid, V_i, c_i, p_i)$ to the copy of $\mathcal{A}$.

  If it later receives $(\textbf{no-block}, sid, V_i, m)$ from $\mathcal{A}$, $\mathcal{S}$ simulates $\mathcal{F}_{\text{KM}}$ receiving this message, and it sends $(\textbf{no-block}, sid, V_i)$ to $\mathcal{F}_{\text{VOTING}}$.
- Upon $\mathcal{A}$ sending $(\textbf{tally}, sid)$ to $\mathcal{F}_{\text{KM}}$, $\mathcal{S}$ lets the simulated $\mathcal{F}_{\text{KM}}$ send the list of stored messages $(\textbf{post}, sid, V_i, c_i, p_i)$ to $A_1, \ldots, A_n$.

  It sends $(\textbf{tally}, sid)$ to $\mathcal{F}_{\text{VOTING}}$ and learns the result.

  Let C be the product of all the c_i's. $\mathcal{S}$ uses the simulation property of the threshold cryptosystem to simulate shares ds_j for the honest A_j's such that C decrypts to the actual result. Furthermore, it also simulates proofs p_j of the shares being correct.
- After $\mathcal{A}$ has delivered both the keys and the messages to honest A_j, $\mathcal{S}$ simulates that A_j sends the decryption share $(\textbf{post}, sid, A_j, ds_j, p_j)$ to $\mathcal{F}_{\text{KM}}$.
- When A_j has received both the public keys and t decryption shares, then $\mathcal{S}$ delivers the $(\textbf{result}, sid, \text{result})$ message from $\mathcal{F}_{\text{VOTING}}$ to $\tilde{A}_j$.

Figure 4: The ideal process adversary $\mathcal{S}$.

Theorem 2 *The voting protocol hybrid* $\pi_{\mathrm{VOTING}}^{\mathcal{F}_{\mathrm{KM}}}$ *securely realizes* $\mathcal{F}_{\mathrm{VOTING}}$ *for the class of non-adaptive adversaries that corrupt less than* t *authorities.*

Proof. We will take a walk one step at a time from the $\mathcal{F}_{\mathrm{KM}}$-hybrid model to the ideal process. In doing so we will use expected polynomial time algorithms and rewind the environment. This is all right as long as we do not do this in the $\mathcal{F}_{\mathrm{KM}}$-hybrid model or the ideal process itself.

Exp$_1$. Define Exp_1 to be the following modification of the $\mathcal{F}_{\mathrm{KM}}$-hybrid model. After $\mathcal{A}$ has submitted the command $(\mathbf{tally}, sid)$ to $\mathcal{F}_{\mathrm{KM}}$ we use the honest authorities' secret shares to decrypt the encrypted votes with valid proofs sent by $\mathcal{A}$ on behalf of corrupt voters. We look at the tapes of the honest voters and if they are not blocked by $\mathcal{A}$, we add their votes to the corrupt voters' votes. This gives us the result of the election.

By the simulation property of the threshold cryptosystem, we may now simulate the honest authorities' decryption shares such that they fit with the result. To do this simulation we do not need knowledge of the honest authorities' secret shares. Using our ability to control the random oracle, we may also simulate proofs of these decryption shares being correct.

$\mathrm{HYB}_{\pi,\mathcal{Z},\mathcal{A}}^{\mathcal{F}_{\mathrm{KM}}} \approx P_1$**.** We define P_1 to be the probability of $\mathcal{Z}$ outputting 1 in Exp_1. It is not possible for $\mathcal{Z}$ to distinguish whether it is running in the $\mathcal{F}_{\mathrm{KM}}$-hybrid model or experiment Exp_1. The result is the same in both cases and indistinguishability follows from the zero-knowledge property of the proofs and the simulation property of the threshold cryptosystem.

We remark that the honest authorities in the zero-knowledge proof choose the challenge as an oracle query of, among other things, an initial message a in the proof. Since a can be chosen from a superpolynomial space $\mathcal{A}$ and $\mathcal{Z}$ cannot guess it beforehand, and therefore they cannot query the oracle about this thing beforehand. For that reason they do not detect that the oracle is being programmed.

Exp$_2$. Define Exp_2 as the following modification of Exp_1. We look at the execution in the interval between key generation having been done and $\mathcal{A}$ not yet having submitted $(\mathbf{tally}, sid)$ to $\mathcal{F}_{\mathrm{KM}}$. After the key generation, we may for each honest voter and each possible vote it can get as input pre-generate the $(\mathbf{message}, sid, c_i, p_i)$ message.

Let A be an algorithm that takes as input the tapes of $\mathcal{A}$, $\mathcal{Z}$ and the pre-generated encrypted votes. It runs the entire execution in this interval, and in the end, it outputs the views of $\mathcal{A}$ and $\mathcal{Z}$. From the views, we may read off the states of $\mathcal{A}$ and $\mathcal{Z}$, restart them, and continue the experiment.

According to Theorem 1 we may replace A with an expected polynomial time algorithm E_A that indistinguishably outputs the same as A, but in addition provides the witnesses for the proofs made by corrupt voters. These witnesses include the votes of these corrupt parties and therefore we do not need to decrypt anything with the honest authorities' secret shares of the private key.

$P_1 \approx P_2$**.** We define P_2 as the probability that $\mathcal{Z}$ outputs 1 at the end of experiment Exp_2. It follows from Theorem 1 that $P_1 \approx P_2$.

Exp_3. Define Exp_3 the following way. Instead of letting the honest voters encrypt their votes and proving in zero-knowledge that the ciphertexts contain correct votes, we let them encrypt 0 and simulate the proofs of correctness. For each possible vote that $\mathcal{Z}$ could give to an honest voter V_i, we construct such a 0-vote and feed A with these ciphertexts and simulated proofs.

$P_2 \approx P_3$. Let P_3 be the probability that $\mathcal{Z}$ outputs 1 after experiment Exp_3. In Exp_3, we still use the real votes to fit the result in the end, and we do not at any point use the honest authorities' shares of the private key. Therefore, by the semantic security of the cryptosystem, the result is the same and $\mathcal{Z}$ cannot distinguish the two experiments. Neither does it allow us to distinguish the views of $\mathcal{A}$ and $\mathcal{Z}$ that A produces, so these transcripts must still look like correct views of $\mathcal{A}$ and $\mathcal{Z}$ acting according to their programs.

Exp_4. We define Exp_4 as a modification of Exp_3 where we go back to using decryption to get $\mathcal{A}$'s votes. Instead of using the votes supplied by E_A, we decrypt the corrupt voters' ciphertexts with valid proofs and use these votes. We may now replace E_A with A since we do not need the votes directly. By definition, A produces valid transcripts of how $\mathcal{A}$ and $\mathcal{Z}$ behave with these inputs and we may therefore replace A with the execution of $\mathcal{A}$ and $\mathcal{Z}$.

$P_3 \approx P_4$. By Theorem 1 we may shift back from E_A to A without being able to tell the difference. Since A produces two good transcripts for how $\mathcal{A}$ and $\mathcal{Z}$ work we may now go back to using $\mathcal{A}$ and $\mathcal{Z}$ also in the interval between key generation and $\mathcal{A}$ submitting $(\mathbf{tally}, sid)$ to $\mathcal{F}_{\mathrm{KM}}$.

$P_4 \approx \mathrm{IDEAL}_{\mathcal{F}_{\mathrm{VOTING}},\mathcal{Z},\mathcal{S}}$. The ideal process and Exp_4 are actually the same experiment. In Exp_4 we submit 0-votes on behalf of honest parties and simulate the proofs, just as $\mathcal{S}$ does. When $\mathcal{A}$ submits $(\mathbf{vote}, sid, V_i, c_i, p_i)$ on behalf of an honest voter we check the proof and decrypt just as $\mathcal{S}$ does. To create something that looks as decryption shares that produce the result we simulate this just as $\mathcal{S}$ does. $\square$

Recycling keys. One could ask whether the keys can be reused for several elections. The security proof fails in this case for the same reasons as described in [SG02] and Footnote 2. The problem is that we can prove non-malleability of the cryptosystem used to encrypt votes but not prove security with respect to general adaptive chosen ciphertext attacks. If we use the same keys in several elections, we give the adversary access to a decryption of the ciphertexts containing the results and therefore an adaptive chosen ciphertext attack. While we see no way to use this attack in practice, we cannot guarantee security.

If we really want to use the keys for several elections that is possible though. We can simply demand that the voter makes a proof of knowledge where votes can be straight-line extracted. For instance, the voter can encrypt votes under a second public key and prove that this has been done correctly. Then votes may be extracted directly from this ciphertext and no rewinding is needed. The authorities tally the votes by stripping away the extra proof and ciphertext and carrying out the usual tallying procedure with the remaining ciphertext.

4 Adaptive adversaries

An adaptive adversary is one that decides during the execution of the protocol which parties to corrupt. After corruption of a party, the adversary may learn some data from earlier computations. To guard against such problems we may specifically specify in protocols that parties should erase certain data. We call this the erasure model. Sometimes the more strict erasure-free security model is preferred. In this model, the party's entire computational history is revealed upon corruption.

The voting schemes are not adaptively secure. The schemes [CGS97, DJ01, BFP$^+$01, DGS03] are in fact not secure in the adaptive setting, even when we allow erasures. Let us sketch a counter-argument for the case of a yes/no election using the scheme in [CGS97] with 2 voters, 3 authorities and a threshold $t = 2$. We refer the reader to [CGS97] for a description of the scheme.

Consider an environment $\mathcal{Z}$ and adversary $\mathcal{A}$, where $\mathcal{A}$ forwards everything it sees to $\mathcal{Z}$ and follows instructions from $\mathcal{Z}$ on how to behave. $\mathcal{Z}$ first asks $\mathcal{A}$ to activate the key-generation step of $\mathcal{F}_{\text{KM}}$ and to deliver all the keys to the relevant parties. Then $\mathcal{Z}$ selects at random that all voters should vote yes or all voters should vote no. It lets the first voter post its vote and then it flips a coin to decide whether to block the second voter or not. If both voters were allowed to post their votes, $\mathcal{Z}$ carries out the entire election according to the protocol. If only the first voter was allowed to post his vote, $\mathcal{Z}$ lets $\mathcal{A}$ activate A_1 to obtain its decryption share. Then it flips a coin and corrupts either A_2 or A_3. From the secret share it obtains it may now compute the result of the election. If everything works out OK, $\mathcal{Z}$ outputs 1. If we are operating in the real-life model everything will work out OK and $\mathcal{Z}$ will output 1 with 100% probability.

To finish the argument we will show that any $\mathcal{S}$ cannot make $\mathcal{Z}$ accept with more than 50% probability. First, $\mathcal{S}$ must provide public keys $g, h = g^s$ for an ElGamal cryptosystem. Second it must provide verification keys $h_1 = g^{s_1}, h_2 = g^{s_2}, h_3 = g^{s_3}$ to the authorities. Here s, s_1, s_2, s_3 may or may not be known to $\mathcal{S}$ and may or may not be chosen according to the protocol. Having given these keys to $\mathcal{Z}$ $\mathcal{S}$ must now produce the vote (x, y) for the first voter. At this point it cannot know the result since if it queries $\mathcal{F}_{\text{VOTING}}$ for the result, then $\mathcal{Z}$ has 50% probability of letting the second voter vote, and then the result will be wrong and $\mathcal{Z}$ will be able to distinguish. From now on, we look at the case where (x, y) has been produced without knowledge of the result, and where this is the only vote to be cast. $\mathcal{S}$ must try to make it look like (x, y) decrypts to the result. First, it must produce a decryption share w_1 for the first authority. Then depending on $\mathcal{Z}$'s coin-flip, it must give either s_2 or s_3 to $\mathcal{Z}$ according to which authority $\mathcal{Z}$ decides to corrupt. To make $\mathcal{Z}$ accept with more than 50% probability, $\mathcal{S}$ must be able to make it look like (x, y) decrypts to the result in both cases. In other words, we have

$$G^{\text{result}} = y/w_1^{\lambda_{1,\{1,2\}}} x^{s_2\lambda_{2,\{1,2\}}} = y/w_1^{\lambda_{1,\{1,3\}}} x^{s_3\lambda_{3,\{1,3\}}},$$

where the Lagrange coefficients are $\lambda_{1,\{1,2\}} = 2, \lambda_{2,\{1,2\}} = -1, \lambda_{1,\{1,3\}} = 3/2, \lambda_{3,\{1,3\}} = -1/2$. This implies that we can compute $w_1 = x^{2s_2 - s_3}$ and $y = G^{\text{result}} x^{3s_2 - 2s_3}$. However, since (x, y)

was chosen before the result was known to $\mathcal{S}$ there is at least 50% probability that $\mathcal{S}$ could not have done this. $\mathcal{Z}$ only has 50% probability of outputting 1 in the ideal process and it can therefore distinguish.

Adaptive security in the erasure model. We can deal with an adversary that may adaptively corrupt voters quite easily. The voters simply erase the plaintext vote and the randomness after they have computed the encrypted vote. This way an adaptive adversary does not learn anything by corrupting a voter. We find the erasure model to be somewhat reasonable since a good voting system should specify that voters delete the randomness and the vote used in order to give some rudimentary receipt-freeness.

To guard against adversaries that adaptively corrupt authorities we can use techniques from [CGJ$^+$99, JL00, LP01]. Let us briefly sketch how to do this. All the homomorphic cryptosystems in [CGS97, BFP$^+$01, DJ01, DGS03] require that in the decryption process we raise the ciphertext C or part of the ciphertext to a secret value s. In the abovementioned schemes we share s using a polynomial f of degree $t-1$, and give each authority a share $s_i = f(i)$. Lagrange interpolation can then be used to perform the decryption. As we saw before, this technique causes trouble in the adaptive setting. However, if we instead use a linear secret sharing of s, i.e., select $s_1, \ldots, s_{n-1}$ at random and $s_n = s - \sum_{i=1}^{n-1} s_i$, then we can cope with an adaptive adversary. To recover if an authority fails to provide its decryption share, we also use polynomials $f_1, \ldots, f_n$ of degree $t-1$ to secret share $s_1, \ldots, s_n$. I.e., $f_i(0) = s_i$ and $s_{i,j} = f_i(j)$. Authority j knows all the shares $\{s_{i,j}\}_{i=1,\ldots,n}$. The verification keys now also include trapdoor commitments, for instance Pedersen commitments, to the $s_{i,j}$'s. In the simulation, we pick all the shares $s_1, \ldots, s_n$ at random. When the first honest authority is about to compute its share, it computes the share such that it fits with the result and all the other authorities' shares, and it simulates a proof of correctness. The authorities have to go through a more complicated protocol to compute the result and anybody wishing to verify the result also has to do more work, but it is still well within what is practical. The voters do not pay any performance penalty when having to use this type of voting scheme instead of the original type of voting scheme, for them the protocol looks the same.

Adaptive security in the erasure-free model. Let us also briefly sketch how to construct a voting scheme that securely realizes the ideal voting functionality against adaptive adversaries in the erasure-free model.

Damgård and Nielsen [DN03] suggest a threshold cryptosystem secure against adaptive adversaries in the erasure-free model, which is based on Paillier encryption [Pai99]. It turns out that their solution is well suited for voting. The idea is to have the public key contain a ciphertext $K = E_{pk}(1; r_K)$. To encrypt a vote v_i, the voter lets $c_i = K^{v_i} E_{pk}(0; r_i)$. By the homomorphic property of Paillier encryption $c_i = E_{pk}(v_i; r_K^{v_i} r_i)$[4]. Taking the product of all the voters' encrypted votes we therefore get a ciphertext C encrypting the result. Now, the authorities can use threshold decryption techniques to decrypt C and get the result. To prove security of the protocol we have to present a simulator $\mathcal{S}$ that fares as well as any adversary $\mathcal{A}$. The idea in the protocol is that $\mathcal{S}$ can choose $K = E_{pk}(0; r_K)$. This enables $\mathcal{S}$ to simulate

---

[4] We use multiplicative notation for the randomness in Paillier encryption.

an encrypted vote by setting $c_i = E_{pk}(0; r_{\mathcal{S},i})$. If the voter is corrupted then $\mathcal{S}$ learns the real vote v_i and must simulate the encryption. It does so by setting $r_i = r_{\mathcal{S},i} r_K^{-v_i}$, which makes $c_i = K^{v_i} E_{pk}(0; r_i)$.

The next question is how a voter should prove that he has submitted a legal vote. In [DN00] a practical scheme for UC commitment and UC zero-knowledge proofs is suggested. The voter can make a UC commitment to v_i and prove in zero-knowledge that v_i is on the correct form and that c_i encrypts v_i. The zero-knowledge proofs of correctness of a vote suggested in [DGS03] are non-erasure Σ-protocols, see [DN03] for a definition, i.e., if we simulate a proof, and later learn the witness, then we can simulate the prover's internal tape. This means that we can also simulate an honest voter's proof of correctness of a vote should he ever be corrupted. These proofs are interactive, however, using the Fiat-Shamir heuristic we can make them non-interactive.

$\mathcal{S}$ faces another problem than simulating honest voters and simulating their tapes as they are corrupted. Suppose the adversary creates an encrypted vote on behalf of one of the corrupt voters. Then $\mathcal{S}$ will have to figure out what to submit to the voting functionality on behalf of said voter. It cannot use the ciphertext of the voter because in the simulation the "encrypted" vote is just an encryption of 0 and does not tell us what v is. However, since the corrupt voter is using a UC commitment scheme $\mathcal{S}$ learns v_i when the commitment is made; the UC commitment scheme is set up such that for any corrupt voter $\mathcal{S}$ can extract the vote v_i.

Unfortunately, the UC commitment scheme from [DN00] is not practical for our purpose. The problem is that each voter needs to receive his own commitment key, and in a large election this implies that we have to generate quite a lot of keys and distribute them to the voters. Using non-malleable commitments, Damgård and Groth [DG03] show how to make it possible to remove this requirement of each voter having his own set of commitment keys. We therefore suggest using their UC commitment scheme instead. This does create a new problem though, since the non-malleable commitments they suggest are not that efficient. To solve this problem we therefore use the simulation-sound trapdoor commitments of [GMY03].

We have now sketched the voters' actions in the protocol. Remaining is to describe how to decrypt the ciphertext C containing the result. Here the idea is that the public key will contain an encryption $R = E_{pk}(0; r_R)$. Using R, the authorities create a rerandomization of C before decryption. In the simulation, however, we use $R = E_{pk}(1; r_R)$ and therefore we can set up the "rerandomization" of C such that we get an encryption of the result. After this, the authorities can carry out a standard threshold decryption protocol to get the result. We refer to [DN03] for a description of how to carry out this rerandomization.

Since the voter has to make a UC commitment to v_i, this protocol is less efficient than the static solutions already proposed in the literature. It is still practical though. The authorities also have to carry out more work, but also from their point of view the protocol is practical.

5 Conclusion

We have proved that the popular type of voting protocols based on homomorphic threshold encryption realizes an ideal voting functionality against non-adaptive adversaries. This implies in particular that the voting scheme satisfies important properties such as privacy, accuracy, robustness and fairness. We have sketched how to modify this type of voting scheme to be secure against adaptive adversaries in both the erasure model and the erasure-free model.

This does not mean that all problems are solved. The universal composability framework treats corruption as an either/or matter. It does not deal with securing the voting machines themselves. Work addressing issues such as incoercibility, receipt-freeness and protection against hackers therefore remains important.

6 Acknowledgment

Thanks to Ivan Damgård for asking whether the schemes based on homomorphic threshold encryption are secure in the universal composability framework.

References

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *proceedings of CRYPTO '02, LNCS series, volume 2442*, pages 417–432, 2002.

[BF97] Dan Boneh and Matthew K. Franklin. Efficient generation of shared rsa keys. In *proceedings of CRYPTO '97, LNCS series, volume 1294*, pages 425–439, 1997.

[BFP$^+$01] Oliver Baudron, Pierre-Alain Fouque, David Pointcheval, Guillaume Poupard, and Jacques Stern. Practical multi-candidate election scheme. In *PODC '01*, pages 274–283, 2001.

[BM03] Mike Burmester and Emmanouil Magkos. Towards secure and practical e-elections in the new era. In D. Gritzalis, editor, *Secure Electronic Voting*, pages 63–72. Kluwer Academic Publishers, 2003.

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

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145, 2001. Full paper available at `http://...3`.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *proceedings of CRYPTO '94, LNCS series, volume 893*, pages 174–187, 1994.

[CGJ$^+$99] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *proceedings of CRYPTO '99, LNCS series, volume 1666*, pages 98–115, 1999.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally eficient multi-authority election scheme. In *proceedings of EUROCRYPT '97, LNCS series, volume 1233*, pages 103–118, 1997.

[DG03] Ivan Damgård and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. In *STOC '03*, pages 426–437, 2003.

[DGS03] Ivan Damgård, Jens Groth, and Gorm Salomonsen. The theory and implementation of an electronic voting system. In D. Gritzalis, editor, *Secure Electronic Voting*, pages 77–100. Kluwer Academic Publishers, 2003.

[DJ01] Ivan Damgård and Mads J. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001, LNCS series, volume 1992*, 2001.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *proceedings of CRYPTO '00, LNCS series, volume 1880*, pages 432–450, 2000.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *proceedings of CRYPTO '03, LNCS series, volume 2729*, pages 247–264, 2003.

[FS01] Pierre-Alain Fouque and Jacques Stern. Fully distributed threshold rsa under standard assumptions. In *proceedings of ASIACRYPT '01, LNCS series, volume 2248*, pages 310–330, 2001.

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *proceedings of EUROCRYPT '99, LNCS series, volume 1592*, pages 293–310, 1999.

[GMY03] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In *proceedings of EUROCRYPT '03, LNCS series, volume 2656*, pages 177–194, 2003. Full paper available at `htpp://eprint.iacr.org/2003/037`.

[JL00] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *proceedings of EUROCRYPT '00, LNCS series, volume 1807*, pages 221–242, 2000.

[LGT$^+$03] Costas Lambrinoudakis, Dimitris Gritzalis, Vassilis Tsoumas, Maria Karyda, and Spyros Ikonomopoulos. Secure electronic voting: The current landscape. In D. Gritzalis, editor, *Secure Electronic Voting*, pages 101–122. Kluwer Academic Publishers, 2003.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constant round secure two-party computation. In *proceedings of CRYPTO '01, LNCS series, volume 2139*, pages 408–432, 2001.

[LP01] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In *proceedings of ASIACRYPT '01, LNCS series, volume 2248*, pages 331–350, 2001.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite residuosity classes. In *proceedings of EUROCRYPT '99, LNCS series, volume 1592*, pages 223–239, 1999.

[SG02] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, 2002.