Cryptographic Protocols FS2011 [1]

Stefan Heule

August 30, 2011

Contents

Part I

Interactive Proofs and Zero-Knowledge Protocols

1 Introduction

1.1 Conventional Proofs

We can first look at formal (conventional) proof systems. Here, the *statements* considered are strings (i.e., finite sequences of symbols) over a finite alphabet, formed according to some syntactic rules. The semantics defines which statements are true, and a proof itself is also a string. The verification algorithm takes as input a statement and a proof, and outputs "true" or "false". For such a proof system, we have two requirements:

- Soundness: Only true statements have proofs.
- Completeness: Every true statement has a proof.

Even though this is often not stated explicitly, we want of course that the verification is *efficient*.

1.2 Interactive Proofs

On the other hand, we can also have interactive proofs, where a prover (often called Peggy) tries to proof a statement to a verifier (often Vic). There are at least three motivations for such proofs:

- Interactive proofs can be *zero-knowledge*.
- Interactive proofs are more powerful than static proofs.
- These proofs are used in several applications, such as digital signature schemes, entity authentication and secure multi-party computation.

There are two types of interactive proofs:

- Proofs of statements, such as "345 has 3 prime factors", or "P=NP".
- Proof of knowledge, e.g., "I know x such that $z = g^x$".

For interactive proofs, the requirements are as follows:

- Completeness: If the statement is true [or the prover known the claimed information], then the proof will be accepted by the verifier.
- Soundness: If the statement is false [or the prover does not know the claimed information], then for all prover strategies the proof will be accepted by the verifier only with *negligible probability*.
- Zero-knowledge: The prover leaks no information to the verifier.

2 Simple Examples

2.1 Graph Isomorphism

The problem of graph isomorphism can be solved using a simple interactive proof, where the prover does not leak any information about the isomorphism to the verifier. If Peggy wants to prove to Vic that two graphs G and H are isomorph, Peggy can first choose a random permutation π and compute the permutation of G, i.e., $T = \pi G \pi^{-1}$. She then sends T to Vic, who sends a (random) challenge bit c to Peggy. If $c = 1$, Peggy replies by providing an isomorphism for T and G, otherwise one for T and H. This protocol is then repeated s times (e.g., $s = 50$).

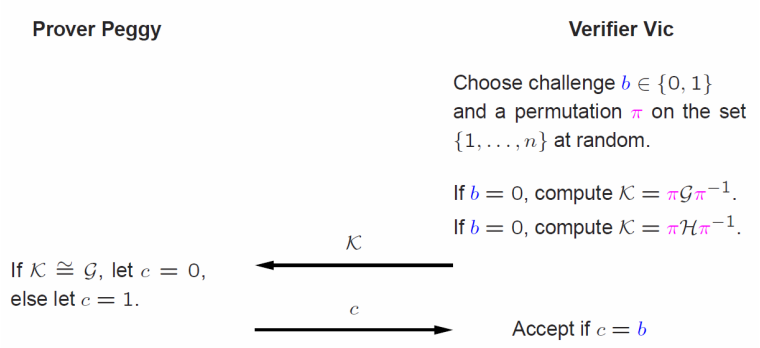

Figure 1: An interactive proof for graph non-isomorphism

Graph non-isomorphism is a statement for which no polynomial algorithm is known. However, we can give an interactive proof, as shown in Figure 1. Note that the verification is indeed efficient. However, the prover has to perform expensive operations.

2.2 Square Roots modulo m: Fiat-Shamir Protocol

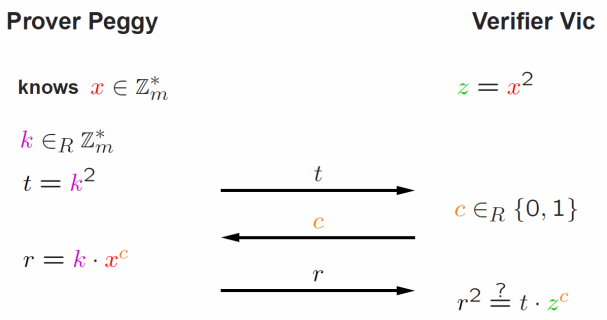

Figure 2: Fiat-Shamir protocol

The protocol depicted in Figure 2 has been proposed by Fiat and Shamir in 1986. Let m be an RSA-modulus. The protocol allows Peggy to prove that she knows at least one square root x (modulo m) of a given number z ($z = x^2 \mod m$). Clearly, this also implies that z is a quadratic residue; the protocol is also a proof of a statement.

If Peggy knows x, then Vic will accept with probability 1. Furthermore, Vic is convinced that Peggy knowns x, since anyone who can answer both challenges 0 and 1 can easily compute x:

$$\begin{aligned} h^r = t \cdot z^c,\ h^{r'} = t \cdot z^{c'} \quad &\Longrightarrow \quad h^{r-r'} = z^{c-c'} = h^{x(c-c')} \\ &\Longrightarrow \quad r - r' \equiv x(c - c') \mod q \\ &\Longrightarrow \quad x \equiv \frac{r - r'}{c - c'} \mod q \end{aligned}$$

If Peggy does not know x, she can at most be prepared to answer one of the two challenges, and will thus be caught with probability $1/2$.

2.3 Higher Roots modulo m: Guillon-Quisquater Protocol

We can also consider a variant of the Fiat-Shamir protocol, as shown in Figure 3. The challenge bit is now chosen from the range $\{0, \ldots, e-1\}$, such that the probability of cheating is now $1/e$ instead of $1/2$. Thus, this protocol is more efficient, as less rounds are needed if e is appropriately chosen. Note that this is no longer a proof of a statement, as for $\gcd(e, \varphi(m)) = 1$, every z has an eth root.

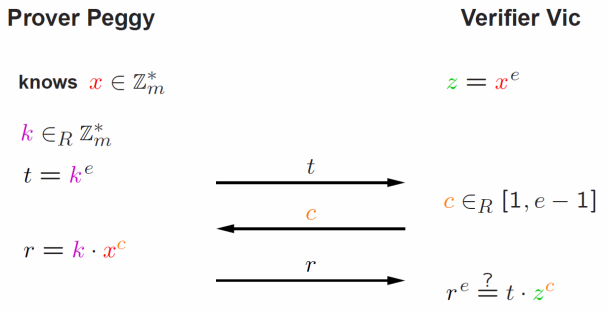

Figure 3: Guillou-Quisquater protocol

2.4 Discrete Logarithm and the Schnorr Protocol

Let G be a group of prime order $|G|$, in which computing the discrete logarithm is hard. Peggy would like to proof that she knows the DL x of an element z (e.g., her public key) with regard to the basis g (i.e., $z = g^x$). Since the group is cyclic, any element has a DL x, that is the protocol is a proof of knowledge only.

2.5 Constructing a Digital Signature Scheme from an Interactive Proof

A digital signature scheme is, by definition, non-interactive. To overcome the problem that there is no verifier that (interactively) provides the challenge bits, we use a hash function. The number of rounds s is chosen such that the list of all k challenges can only be guessed with negligible probability. Then, the non-interactive proof works as follows:

1. Peggy generates for $i = 1, \ldots, s$ the first message t_i of the ith round.

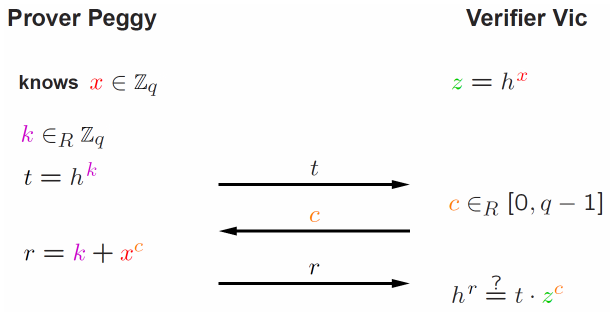

Figure 4: Schnorr protocol

2. Peggy generates the s challenges by using a cryptographically secure hash function h:

$$[c_1, \ldots, c_s] = h(t_1, \ldots, t_s, m)$$

   The result depends on the message m itself, and on all t_i.

3. Peggy generates the s answers $r_1, \ldots, r_s$ for the given challenges.

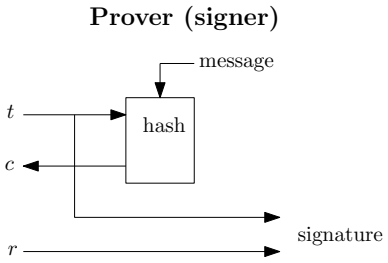

Figure 5: Fiat-Shamir heuristic.

The system is illustrated in Figure 5. Intuitively, Peggy cannot influence the challenge bits $c_1, \ldots, c_s$ (enough), but has to be prepared to answer any challenge. This idea is called Fiat-Shamir heuristic, and as the name suggests, its security is only heuristic; it is not proven in terms of a well-defined hard problem. In practice however, this approach is very useful, as it provides a highly efficient digital signature scheme, and various alternatives for the underlying hard problem are possible.

For such a system to be secure, the challenge space should be large.

3 Interactive Proofs of Statements

Definition 3.1 (Interactive Proof)**.** *A pair (P, V) of programs is an* interactive proof *for a formal language L such that*

- ***Completness****. V accepts for all $x \in L$ with probability larger or equal $3/4$.*
- ***Soundness****. V accepts for all $x \notin L$ and all programs P' (instead of P) with probability at most $1/2$.*

The verifier V must run in polynomial time, while P (and P') can be arbitrary programs.

Note that the constants $1/2$ and $3/4$ are chosen rather arbitrarily and can be replaced with other values, as the protocols can be repeated independently.

Definition 3.2. *The set of languages that have an interactive proof is called* **IP**.

We can see **NP** as the set of languages for which a non-interactive proof exists. Thus,

$$\mathbf{NP} \subseteq \mathbf{IP}$$

Theorem 3.1. IP = PSPACE

4 Zero Knowledge

In a zero-knowledge protocol, the verifier should not obtain any information about the secret value of the prover. Even more, the verifier should not obtain *any* information, which he did not possess already before. To formalize what it means for a program to obtain knowledge, the following elegant and in security definition often encountered idea is used. A verifier does not obtain any knowledge during the protocol, if he could generate the complete communication efficiently by itself, without communicating with the prover. This is formalized using a simulator, which interacts with the verifier to efficiently simulate the complete communication, indistinguishably from a real protocol run with the verifier.

Definition 4.1 (Zero Knowledge)**.** *A protocol (P, V) is called (black box)* zero-knowledge (ZK) *if there exists an efficient simulator S with access to a (possibly) dishonest efficient verifier V' such that for every V' it outputs a simulated transcript T' which is indistinguishable from the real transcript T. This is shown in Figure 6.*

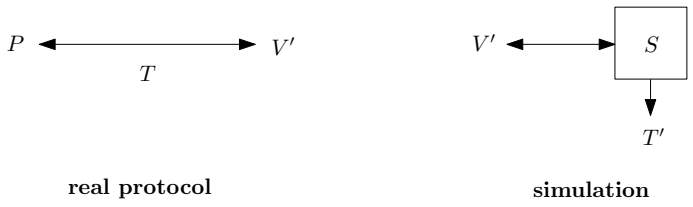

Figure 6: Zero knowledge; the simulator.

A protocol is called *honest-verifier zero-knowledge (HVZK)*, if the simulator exists for the (correct) verifier V (but possibly not for other verifiers).

4.1 Perfect Zero-Knowledge Proof

Definition 4.2 (c-simulatable)**.** *A three-move protocol round with challenge space $\mathcal{C}$ is* c-simulatable *if for any value $c \in \mathcal{C}$ one can efficiently generate a triple (t, c, r) with the same distribution as occurring in the protocol (conditioned on the challenge being c).*

Note that all previously discussed three-move protocols are c-simulatable. Also, a c-simulatable round is always HVZK, as the simulator for the honest verifier just chooses a $c \in \mathcal{C}$ at random and generates t and r according to the c-simulation.

Theorem 4.1. *A protocol consisting of independent perfect HVZK (e.g., c-simulatable) rounds $(T_1, C_1, R_1), \ldots, (T_k, C_k, R_k)$ is perfect HVZK. Is the challenge space $\mathcal{C}$ additionally a polynomially bounded (per round) and the challenge uniformly chosen (by an honest verifier), then it is also perfect zero-knowledge.*

4.2 Proofs of Knowledge

Knowledge w is formalized with regard to a given string z (the input of both Vic and Peggy) and an efficiently computable predicate

$$Q : \{0,1\}^* \times \{0,1\}^* \to \{\texttt{true}, \texttt{false}\}$$

The goal of Peggy is to prove knowledge of some string w such that $Q(z,w) = \texttt{true}$. It is possible that multiple such w exists; it suffices for Peggy to know any one of them. Often, the existence of w is clear from the context. A proof of knowledge is only sensible if Peggy's resources are (polynomially) limited.

Definition 4.3 (Proof of Knowledge)**.** *An interactive protocol between P and V is a* proof of knowledge for the predicate Q, *if*

- *Completeness. V accepts with probability 1 if the protocol has been carried out correctly, and if P knows w such that $Q(z,w) = \texttt{true}$.*
- *Correctness. There exists an efficient program called the* knowledge extractor K, *which can interact with* any *program P' for which V accepts with non-negligible probability, and outputs a valid secret x.*

Definition 4.4 (Two-Extractable)**.** *A three-move protocol round with challenge space $\mathcal{C}$ is* two-extractable *if from any two triples (t,c,r) and (t,c',r') with $c \neq c'$ accepted by Vic one can efficiently compute w with $Q(z,w) = \texttt{true}$.*

All previously discussed protocols fulfill this property.

Theorem 4.2. *An interactive protocol consisting of s two-extractable rounds with challenge space $\mathcal{C}$ is a proof of knowledge for the predicate Q if $1/|\mathcal{C}|^s$ is negligible (in the input size $|z|$).*

Proof. Let z be a concrete string, and let p be the (non-negligible) probability that V accepts for P' (as in Definition 4.3). We construct a knowledge-extractor that finds a correct string w.

If P' is probabilistic, we can model its randomness by some bitstring l. P' is deterministic for any fixed l The mean of the success probability over all l is p. The knowledge-extractor is defined as follows:

1. Choose l at random.
2. Generate two independent protocol runs between P' and V (with the fixed bits l).
3. If in the second step V accepts in both runs and has chosen distinct challenge sequences, then we consider the first round (of the s rounds) in which the two challenge sequences differ. From the two corresponding triple, we extract w (using the two-extractable property of the protocol).

It is left to show that the knowledge-extractor is efficient. Let $f(l)$ be the success probability of P' for randomness l. Let L be the corresponding random variable, thus $\mathbb{E}[L] = p$. The probability that the knowledge-extractor generates two accepting protocol runs is thus $f(l)^2$ (ignoring the very small probability that W might generate two times the same l). Thus, the expected number of rounds until W generates two accepting runs is $\mathcal{O}\left(\mathbb{E}\left[f(L)^2\right]\right)$. Using Jensen's equality, we get

$$\mathbb{E}\left[f(L)^2\right] \geq \mathbb{E}\left[f(L)\right]^2 = p^2$$

Thus, the runtime of the knowledge-extractor is efficient with $\mathcal{O}\left(1/p^2\right)$. □

4.2.1 Witness Hiding and Witness Independence

Definition 4.5 (Witness Hiding)**.** *A proof of knowledge (P, V) is called* witness-hiding *if there exists no efficient algorithm V' which, after interactivity arbitrarily with P (possibly in many protocol instantiations), can play the role of P and make V accept with non-negligible probability.*

For a predicate Q and value z, let $\mathcal{W}_z = \{w \mid Q(z, w) = \texttt{true}\}$ be the set of witnesses for z. Consider a setting where $\mathcal{W}_z \geq 1$.

Definition 4.6 (Witness Independence)**.** *A proof of knowledge (P, V) for predicate Q is called* witness independent *if for any verifier V' the transcript is independent of which witness the prover is using in the proof.*

Theorem 4.3. *If one can generate a pair (w, z) with w uniform in $\mathcal{W}_z$ and it is computationally infeasible to find a triple (z, w, w') with $w \neq w'$ and $w, w' \in \mathcal{W}_z$, then every witness-independent proof of knowledge for Q is witness-hiding.*

Proof. One can generate a pair (z, w) (as above) at random and observe the protocol between P and V. Suppose there were a verifier V' that makes V accept (by first interacting with P). Therefore, we can use the witness-extractor W to extract w'. Since w has been chosen uniformly at random from $\mathcal{W}_z$ and because of the witness-independence property, with non-negligible probability we have $w \neq w'$. This is a contradiction that we cannot find a triple (z, w, w'). □

5 Bit Commitments

5.1 Hamiltonian Cycle

We first look at an example to introduce bit commitments informally, and then define them more rigorously later on. Suppose Peggy wants to convince Vic that a graph G has a Hamiltonian cycle. This can be done with the following protocol, which is repeated s times. It is also shown in Figure 7.

1. Peggy generates a random permutation σ and the corresponding permuted graph $H = \sigma G \sigma^{-1}$. Peggy then writes down the adjacency matrix, but uses opaque duct tape to cover all n^2 entries.
2. Vic chooses (randomly) a challenge bit c. If $c = 0$, then Peggy opens all entries and provides the permutation σ. Otherwise, i.e., if $c = 1$, Peggy will only open the entries that correspond to the Hamiltonian cycle.
3. Vic verifies that indeed $H = \sigma G \sigma^{-1}$ if $c = 0$. Otherwise, he can verify that the entries opened by Peggy form a Hamiltonian cycle.

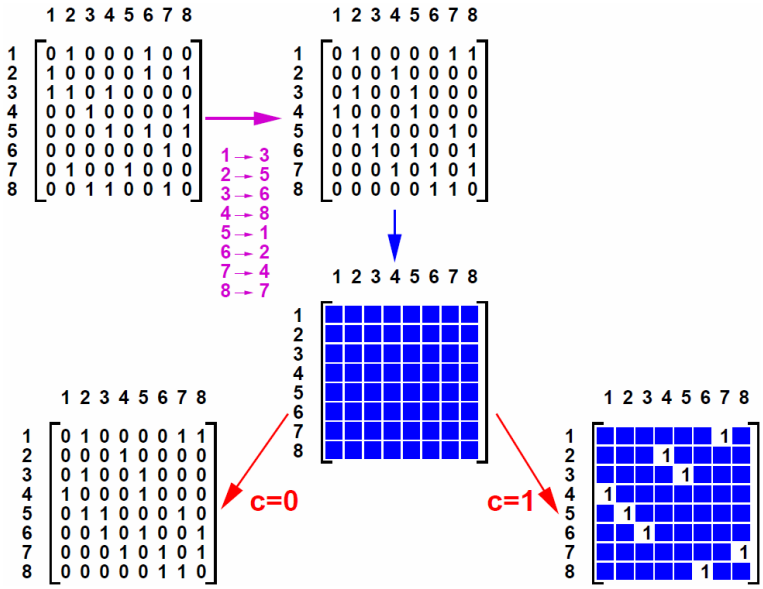

Figure 7: Zero-knowledge protocol for Hamiltonian cycles.

As with other protocols that we have seen earlier, a cheating prover can prepare for either one of the challenges, but not for both. The only open problem remains how we can realize "duct tape" digitally.

5.2 Definition

Definition 5.1 (Bit Commitment Scheme)**.** *A* bit commitment scheme *consists of a pair of protocols* Commit *and* Open *between a prover P and a verifier V. For every instance of* Commit*, there exists a corresponding instance of* Open*. An execution of both protocols can either be*

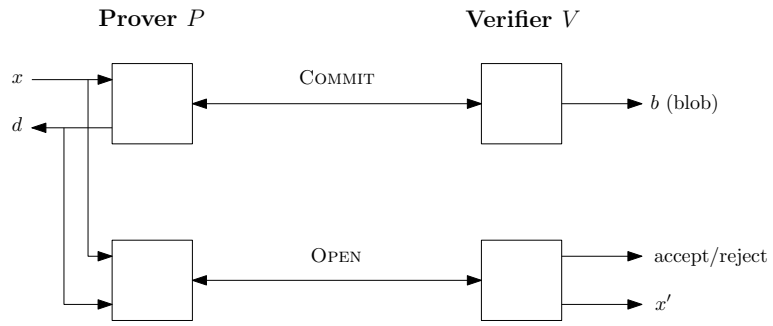

Figure 8: Visualization of bit commitments.

accepted or rejected by V. P has one input x for Commit *and V receives a value b (the* blob*, or the* commitment*) and P receives a value d. For* Open*, P has the input (x, d). V on the other hand has the input b and receives an output x'. We have the following requirements:*

- ***Correctness.*** *If both P and V act according to the protocols, then V accepts both for* Commit *and* Open*. Furthermore, $x' = x$.*
- ***Hiding.*** *No cheating V' can obtain information about the committed value, unless* Open *is executed.*
- ***Binding.*** *For any cheating P', if V accepts* Open*, there is at most one value x which P can open.*

There exist two types of security for bit commitment schemes.

- **Type H.** The hiding property is secure in an information-theoretic sense, i.e., all possible values for x result in the same probability distribution for b. In particular, the blob b is statistically independent from x.
- **Type B.** The binding property is secure in an information-theoretic sense, i.e., x is determined uniquely by b. There exists no values x and $x' \neq x$ and d, d' such that V accepts for both (x, b, d) as well as (x', b, d').

Note that a commitment scheme cannot be both of type H and B at the same time.

5.3 Bit Commitment Scheme of Type H

Let G be a cyclic group of order q with two generators g and h, where the discrete logarithm of h to the basis g is assumed not to be known. Peggy can commit to a value $x \in \{0, \ldots, q-1\}$, by choosing r randomly from $\{0, \ldots, q-1\}$ with blob b as follows

$$b = g^x h^r$$

The commitment can easily be opened by sending x and $d = r$. Due to the random choice of r, b is a random element from G for every value of x. However, Peggy could cheat if she knows $\log_g h$ in G (or is able to compute it).

5.4 Quadratic Residue

Before we give an example of a bit commitment scheme of type B, we first provide some background information on quadratic residues.

Definition 5.2 (Quadratic Residue)**.** *An integer q is called a* quadratic residue *modulo n if it is congruent to a perfect square* $\pmod n$*. That is, if there exists an integer x (called square root modulo n) such that*

$$x^2 \equiv q \mod n$$

Otherwise, q is called a quadratic nonresidue $\pmod n$*.*

We use $\mathbf{QR}_m$ and $\mathbf{QNR}_m$ for the set of quadratic residues and quadratic nonresidues, respectively.

Theorem 5.1. *For every odd prime p it holds that $|\mathbf{QR}_p| = |\mathbf{QNR}_p| = (p-1)/2$ and every $a \in \mathbf{QR}_p$ there are exactly two square roots modulo p.*

Definition 5.3 (Legendre Symbol)**.** *For an odd prime p, we define the* Legendre symbol *of a modulo p as follows*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 \text{ if } a \text{ is a quadratic residue modulo } p \text{ and } a \not\equiv 0 \pmod p & a \in \mathbf{QR}_p \\ -1 \text{ if } a \text{ is a quadratic non-residue modulo } p & a \in \mathbf{QNR}_p \\ 0 \text{ if } a \equiv 0 \pmod p. \end{cases}$$

Theorem 5.2. *For an odd prime p it holds that*

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \pmod n$$

Definition 5.4 (Jacobi Symbol)**.** *For an integer $m = p_1 \cdots p_r$, where p_i are (not necessarily distinct) prime numbers, we define the Jacobi symbol as*

$$\left(\frac{a}{m}\right) = \left(\frac{a}{p_1}\right) \cdots \left(\frac{a}{p_r}\right)$$

We are now interested in RSA moduli of the form $m = pq$. An integer a is a quadratic residue modulo m if and only if this is the case modulo p and modulo q:

$$a \in \mathbf{QR}_m \quad \Longleftrightarrow \quad (a \in \mathbf{QR}_p) \wedge (a \in \mathbf{QR}_q)$$

This, a fourth of the numbers in $\mathbb{Z}_m^*$ are quadratic residues, and for all such numbers a we have $\left(\frac{a}{m}\right) = 1$.

Definition 5.5 (QR-Problem)**.** *Let $m = pq$ be a RSA modulus with unknown factors. The* quadratic residuosity problem *is, for a given a with $\left(\frac{a}{m}\right) = 1$ to decide whether a is a quadratic residue.*

There is no algorithm known that is (significantly) more efficient than to factorize m.

5.5 Bit Commitment Scheme of Type B

Let $m = pq$ be an RSA modulus, and t a quadratic nonresidue modulo m with $\left(\frac{t}{m}\right) = 1$. For instance, these parameters could be generated by a trusted party, or by Peggy (in this case, Peggy will have to prove to Vic that t is indeed a quadratic nonresidue).

To commit to a bit $x \in \{0, 1\}$, Peggy chooses a random element r from $\mathbb{Z}_M^*$ and computes the blob as $b = r^2 t^x$. This scheme is of type B, as $x = 0$ if and only if $b \in \mathbf{QR}_m$; x is completely fixed with b. However, Vic could cheat by solving the QR-problem.

6 Unifying Zero-Knowledge Proofs of Knowledge

This section is based on [Mau09], and contains all important definitions, theorems and proofs (often without changes).

6.1 One-Way Group Homomorphisms

We consider two groups $(G, \star)$ and $(H, \otimes)$. We assume that the group operations $\star$ and $\otimes$ are efficiently computable. A function $f : G \to H$ is a homomorphism if

$$f(x \star y) = f(x) \otimes f(y)$$

We consider the case where f is (believed to be) a one-way function, such that it is infeasible to compute x from $f(x)$ for a randomly chosen x. In this case it is meaningful for a prover Peggy to prove that she knows an x such that for a given value z we have $z = f(x)$. To simplify the notation we write $[x]$ instead of $f(x)$. We can consider $[x]$ to be an embedding of $x \in G$ in H. Given two embedded values $[x]$ and $[y]$ we can efficiently compute

$$[x \star y] = [x] \otimes [y]$$

without knowing x or y, due to the homomorphism.

6.1.1 The Main Protocol

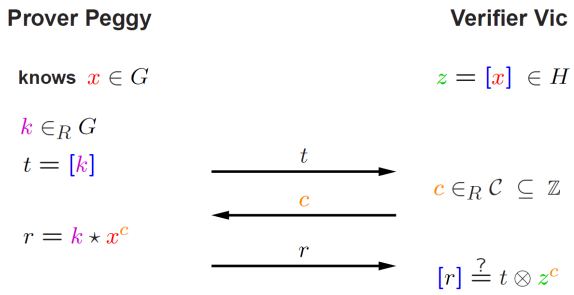

Figure 9: General protocol of knowledge, for a given value z, of a value x such that $z = [x]$, where $x \mapsto [x]$ is a (one-way) group homomorphism.

The protocol in Figure 9 is a proof of knowledge of a value x such that $z = [x]$, for a given value z, provided that the two condition of the following theorem are satisfied. If the challenge space $\mathcal{C}$ (which can be an arbitrary subset of $\mathbb{N}$) is small, one needs several (three-move) rounds to reduce the soundness error to be negligible.

Theorem 6.1. *If value $l \in \mathbb{Z}$ and $u \in G$ are known such that*

1. $\gcd(c_1 - c_2, l) = 1$ *for all $c_1, c_2 \in \mathcal{C}$ (with $c_1 \neq c_2$), and*
2. $[u] = z^l$,

then the thee-move protocol round described in Figure 9 is 2-extractable. Moreover, a protocol consisting of s rounds is a proof of knowledge if $1/|\mathcal{C}|^s$ is negligible, and it is zero-knowledge if $|\mathcal{C}|$ is polynomially bounded.

Proof. 2-extractability can be proven as follows: From r and r' such that $[r] = t \otimes z^c$ and $[r'] = t \otimes z^{c'}$ for two different challenges c and c' we can obtain $\tilde{x}$ satisfying $[\tilde{x}] = z$, as

$$\tilde{x} = u^a \star (r'^{-1} \star r)^b$$

where a and b are computed using Euclid's extended gcd-algorithm such that

$$\ell a + (c - c')b = 1$$

We make use of

$$[r'^{-1} \star r] = [r'^{-1}] \otimes [r] = z^{-c'} \otimes t^{-1} \otimes t \otimes z^c = z^{-c'} \otimes z^c = z^{c-c'}$$

to see that $[\tilde{x}] = z$:

$$\begin{aligned}
[\tilde{x}] &= [u^a \star (r'^{-1} \star r)^b] \\
&= [u]^a \otimes [r'^{-1} \star r]^b \\
&= (z^\ell)^a \otimes (z^{c-c'})^b \\
&= z^{\ell a + (c-c')b} = z
\end{aligned}$$

□

6.2 Special Cases of the Main Protocol

6.2.1 Schnorr's Protocol

The Schnorr protocol is the special case where $(G, \star) = (\mathbb{Z}_q, +)$ (with addition modulo q in $\mathbb{Z}_q$) and H is a group of order q with the group operations written as multiplication. The (one-way) group homomorphism is defined by

$$G \to H : x \mapsto [x] = h^x$$

The challenge space $\mathcal{C}$ can be an arbitrary subset of $[0, q-1]$. The two conditions of Theorem 6.1 are satisfied for $\ell = q$ (if q is prime) and $u = 0$. Note that $[u] = [0] = 1 = z^\ell$ since every element of H raised to the group order $|H| = q$ is the neutral element of H.

6.2.2 Guillou-Quisquater Protocol

The GQ protocol is the special case where $(G, \star) = (\mathbb{Z}_m^*, \cdot) = (H, \otimes)$. The one-way homomorphism is defined by

$$G \to H : x \mapsto [x] = x^e$$

The challenge space $\mathcal{C}$ can be an arbitrary subset of $[0, e-1]$, provided e is prime. The two conditions of Theorem 6.1 are satisfied for $\ell = e$ and $u = z$.

6.2.3 Proof of Correctness of Diffie-Hellman Keys

Let H be a group with prime order $|H| = q$ and generator h used in the Diffie-Hellman protocol. As for the Schnorr protocol, we define a homomorphic embedding by

$$G \to H : x \mapsto [x] = h^x$$

Recall that in the Diffie Hellman protocol, Alice chooses an $a \in \mathbb{Z}_q$ and sends $[a] = h^a$ to Bob and, symmetrically, Bob chooses $b \in \mathbb{Z}_q$ and sends $[b] = h^b$ to Alice. The common secret key is $[ab] = h^{ab}$. It is believed for general groups it is computationally hard to decide whether or not a given key $K \in H$ is the correct key, i.e., whether $K = h^{ab}$ This is known as the Decisional Diffie-Hellman (DDH) problem. For example, if a very powerful organization were willing to compute Diffie-Hellman keys as a commercial service (returning $[ab]$ when given $[a]$ and $[b]$), then the customer could not verify that the key is correct. In this context, as well as in other contexts, it is useful to be able to prove correctness of a Diffie-Hellman key in zero-knowledge, in particular without leaking any information about a or b. This is again achieved by a simple instantiation of the main protocol.

Let $A = [a]$, $B = [b]$ and $C = [c]$ be given. We wish to prove that $c = ab \pmod q$, i.e., that A, B, and C form a so-called Diffie-Hellman triple. For this purpose we define the following one-way group homomorphism which we denote by $[\![\cdot]\!]$ and which is defined in terms of the homomorphism $[\cdot]$ and of B:

$$\mathbb{Z}_q \to H \times H : \; x \mapsto [\![x]\!] = ([x], [xb]) = (h^x, B^x)$$

Note that $[xb]$ can be computed efficiently from $B = [b]$ and x without knowing b. This yields, as a special case of the main protocol for the homomorphism $x \mapsto [\![x]\!]$, the desired proof: One proves knowledge of a preimage x (namely $x = a$) such that

$$[\![x]\!] = (A, C).$$

Due to the particular choice of the homomorphism, this implies $c = ab$.

Part II
Multi-Party Computation

7 Introduction

Multi-party computation is concerned with the problem of some kind of computation that is performed by a number of persons or entities, which do not necessarily trust each other. Entities can attempt to cheat, but the protocol has to detect and prevent this.

Such cooperation problems can easily be solved by a *trusted entity* which performs the computation, and is, as the name suggests, trusted by all other entities in the system. The goal of MPC is thus to *simulate* such a trusted party.

One particular example of an MPC computation is the *secure function evaluation*, where multiple players have a (secret) input and want to compute a specified function of these values. Every player should learn the result, but nothing more.

Definition 7.1 (Adversary)**.** *We consider two types of adversaries:*

- ***passive** adversaries execute the protocol correctly, but try to find information about the secret inputs of the other players.*
- ***active** adversaries can execute sequence of actions, following the protocol or not, with the goal of (collectively) distort the result of the computation and/or obtaining secret information.*

Often we take the viewpoint of a single (central) opponent that controls a certain number of players (actively or passively). As in the previous part, we distinguish between limit and unlimited computational power of adversaries and the security achieved can either be computational or information-theoretical.

Definition 7.2 (Communication)**.** *We make the following assumption about communication:*

- *The players can pairwise communicate securely.*
- *Sometimes we assume the existence of a* broadcast channel*, which usually allows us to achieve better results. A broadcast channel ensures that all players receive the same message.*

Definition 7.3 (Security Requirements)**.** *We have the following security requirements for MPC protocols:*

- ***Privacy.** Any cheating player should not receive information about the inputs of any other player, except what is already known by the input of the cheating player, and the output of the computation.*
- ***Correctness.** Cheating players cannot influence the output of any honest player.*
- ***Fairness.** We call a protocol fair, if the cheating players cannot stop the computation with an advantage at any point. Put another way, if the cheating players receive information about the result, then the honest players receive the complete result.*

- ***Robustness.*** *A protocol is robust if it is not possible to abort it. Once the inputs have been "put" into the protocol (similar to a commitment), then all players can execute the protocol, even if some players try to prevent this.*

Formally we define the security of a MPC protocol with regard to a *specification* between players and a trusted third party (TTP), where the TTP usually preforms the actual computation. A MPC protocol is secure with regard to a specification if the cheating player can only achieve things that they could also do in the specification.

The following table summarizes known results for n player, where t players are dishonest.

Setting	**Adv. type**	**Condition**
cryptographic	passive	$t < n$
cryptographic	active	$t < n/2$
information-theoretic	passive	$t < n/2$
information-theoretic	active	$t < n/3$
i.t. with broadcast	active	$t < n/2$

7.1 Computing the Sum

The following protocol computes the sum (or average) of the n inputs of all players. It is secure for up to $n-1$ passive adversaries.

1. Every player P_i chooses n random values $r_{i1}, \ldots, r_{in}$ such that
$$x_i = r_{i1} + \cdots + r_{in}$$
2. Every players P_i sends r_{ij} to P_j.
3. Every player computes the sum $R_j = \sum_{i=1}^{n} r_{ij}$ of the received values and sends it to all other players.
4. The result of every player is
$$S = \sum_{j=1}^{n} R_j = \sum_{j=1}^{n} \sum_{i=1}^{n} r_{ij} = \sum_{i=1}^{n} x_i$$

7.2 Lagrange Interpolation

Given a set of $k+1$ data points (x_i, y_i) where no two x_i are equal, we can reconstruct the polynomial of degree k that goes through all these points by the interpolation polynomial:

$$L(x) = \sum_{j=0}^{k} y_j l_j(x)$$

of the Lagrange basis polynomials

$$l_j(x) = \prod_{m \neq j} \frac{x - x_m}{x_j - x_m}$$

Note, that

$$l_{j \neq i}(x_i) = 0 \qquad \text{and} \qquad l_i(x_i) = 1$$

8 Oblivious Transfer and Two-Party Protocols

One of the most primitive MPC protocols is *oblivious transfer* (OT) and exists in several variants.

Definition 8.1 (Bit-OT)**.** *Player A has an input bit b and B does not have an input. A does not have an output, and the output of B is an indicator bit (randomly chosen by the protocol without influence from A or B), and the bit b if $c = 1$.*

Intuitively, B receives the bit b with probability 1/2, and otherwise learns nothing. A on the other hand does not know which case happened.

Definition 8.2 (1-2-OT)**.** *The 1-out-of-2 bit-OT is a variant where A has two input bits b_0 and b_1, and B can choose a selection bit c. A does not get an output, and B receives b_c, but no information about b_{1-c}.*

Further variations allow strings instead of bits, or allow multiple (more than two) inputs for A; the so-called 1-out-of-k String-OT.

8.1 Cryptographic 1-out-of-2 String-OT

A chooses the parameter of two RSA systems, $[m_0 = p_0q_0, e_0]$ and $[m_1 = p_1q_1, e_1]$, where m_0 and m_1 are almost identical. B chooses r randomly from $\mathbb{Z}_{m_c}$ (where c is the selection bit) and computes

$$y = r^{e_c} \pmod{m_c}$$

A computes $x_0 = y^{d_0} \pmod{m_0}$ and $x_1 = y^{d_1} \pmod{m_1}$ and sends $u_0 = x_0 + s_0$ and $u_1 = x_1 + s_1$, where s_0 and s_1 are the input strings of A. B computes $s_c = u_c - r$.

This realisation of 1-out-of-2 String-OT is (probably) computationally secure for a passive opponent A, and an actively cheating B.

8.2 Cryptographic 1-out-of-k String-OT

For a power of 2, $k = 2^t$ we can implement 1-out-of-k String-OT using 1-out-of-2 String-OT as follows. A chooses t random pairs of strings $(s_{10}, s_{11}), \ldots, (s_{t0}, s_{t1})$. We define k keys $K_{00\ldots0}, \ldots, K_{11\ldots1}$ by concatenating the corresponding string. All input strings are encrypted with these keys, and we send the cipher texts to B. Also, we send every pair of string to B via 1-out-of-2 String-OT. Thus, B gets exactly one key and can decrypt exactly one of the k values.

8.3 Computing Arbitrary Functions for Two Players

Given a function $f : \{0,1\}^s \times \{0,1\}^t \to \{0,1\}^u$ and inputs x_A and x_B, we proceed as follows to compute $f(x_A, x_B)$. A sends to B via 1-out-of-k String-OT the $k = 2^t$ function values $f(x_A, \cdot)$. B can then choose with index x_B to get the result and send the result to A.

9 Problem Definition and Secret-Sharing

We consider the case of n players $P_1, \ldots, P_n$ that want to compute the result of some function over a field $GF(q)$. In a field, any function can be written in terms of addition and multiplication, which will simplify our task. Typically, a protocol consists of three phases:

- **Input.** The input of every player is distributed among all the other players using secret-sharing.
- **Computation.** Every operation of the computation is performed while preserving the following *invariant*: every intermediate result is distributed among all players (using secret sharing) in a way that the honest players can obtain the correct value, while any number of the dishonest players cannot.

  Addition (and any linear operation in general) will be trivial to support and will not require communication. The more difficult problem will be multiplication.
- **Output.** If a player should receive a value as output, then all players send their shares to this player.

9.1 Secret-Sharing

Secret sharing can be used to distribute a certain secret value s among several entities $P_1, \ldots, P_n$ such that only a qualified subset of the persons can reconstruct the value s.

Definition 9.1. *Let $\mathcal{P}$ be the set of players and $2^{\mathcal{P}}$ its power set. A* access structure (Zugriffstruktur in German) Γ *is a monotone subset, i.e.,*

$$\Gamma \subseteq 2^{\mathcal{P}} \qquad \text{where} \quad \mathcal{M} \in \Gamma \wedge \mathcal{M} \subseteq \mathcal{M}' \implies \mathcal{M}' \in \Gamma$$

Often, Γ is specified using minimal subsets.

Definition 9.2 (Secret-Sharing Scheme)**.** *A secret-sharing scheme for $\mathcal{P}$ and Γ consists of a pair of two efficient protocols:*

- Share *is used to distribute a value s.* Share *is probabilistic and uses secret channels for the distribution of the shares.*
- Reconstruct *is used to (later) reconstruct s. It takes the description of a qualified set $M \in \Gamma$ as well as the shares of all players in this set, and reconstructs s.*

A secret-sharing scheme is called *perfect* if the reconstruction of s succeeds always (there is no error probability, not even a small one). It is further called *ideal* if the shares are from the same domain as s.

For instance, if $\Gamma = \{\mathcal{P}\}$, then only all players together can reconstruct the value. We can use the scheme introduced in 7.1.

9.1.1 Linear Secret-Sharing Schemes

A secret-sharing scheme is called *linear* (over a certain field) if the shares $s_1, \ldots, s_n$ of the players can be computed as a linear function from s and a number of random values $r_1, \ldots, r_m$. That

is, it exists a (constant) $n \times (m+1)$ matrix A, such that

$$\begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} = \begin{pmatrix} A_{10} & A_{11} & \cdots & A_{1m} \\ \vdots & & \ddots & \vdots \\ A_{n0} & A_{n1} & \cdots & A_{nm} \end{pmatrix} \cdot \begin{pmatrix} s \\ r_1 \\ \vdots \\ r_m \end{pmatrix}$$

Linear sharing schemes have the important and useful property that from a sharing of a and a sharing of b one can easily compute a sharing for $a+b$ (all players just add their shares of a and b).

9.1.2 Shamir's Secret Sharing

In a (k, n) threshold scheme, any group of k or more players can reconstruct the secrete, but no other group can. Thus,

$$\Gamma = \{\mathcal{M} \subseteq \mathcal{P} \mid |\mathcal{M}| \geq k\}$$

We consider variables from the finite field $GF(q)$ where every player P_i gets assigned a fixed and commonly known element $\alpha_i \neq 0$. All α_i must be different, thus $q > n$. The dealer chooses $k-1$ random values $a_1, \ldots, a_{k-1}$ from $GF(q)$ and uses the following polynomial to compute the shares:

$$a(x) = s + a_1 x + \cdots + a_{k-1} x^{x-1}$$

Player P_i gets the share $a(\alpha_i)$, and any k shares allow one to reconstruct the polynomial fully, and thus find s as $a(0)$ (e.g., using Lagrange interpolation).

No group of $k-1$ (or less) players can obtain information about s, which can be shown as follows. The secret s can also be seen as a share (for the value $x = 0$). For the $k-1$ shares of the players, any choice of s is compatible and determines the polynomial uniquely. Every value is possible, and the probability distribution for s is the same before and after knowing the $k-1$ shares.

Shamir's secret sharing scheme is linear with

$$A = \begin{pmatrix} a_1^0 & a_1^1 & \cdots & a_1^{k-1} \\ a_2^0 & a_2^1 & \cdots & a_2^{k-1} \\ \vdots & & \ddots & \vdots \\ a_n^0 & a_n^1 & \cdots & a_n^{k-1} \end{pmatrix}$$

10 Information-Theoretically Secure MPC with Passive Adversaries

Theorem 10.1. *A set of n players can compute any function information-theoretically secure against t passive adversaries if and only if $t < n/2$.*

10.1 Impossibility for $n/2$ Adversaries

We will first show that the bound $n/2$ is optimal if we want to compute arbitrary functions. If we only consider linear functions, then (as we have seen) we can tolerate up to $n-1$ passive cheaters.

We consider the case of $n = 2$ and $t = 1$ first, and generalize later. We show that A and B cannot compute the AND of their inputs without either A or B gaining knowledge about the input of the other. For a contradiction, assume that there is such a protocol, defined by program π_A and π_B. Both can be probabilistic and we model this by having another input r_A (and r_B), a random bit string. Now, A and B perform the protocol and both receive the transcript T (the set of all messages exchanged), which will depend on $x - A$, x_B (inputs of A and B), as well as r_A and r_B. Due to privacy, T cannot contain information about x_A if $x_B = 0$, and because of correctness, T must contain full information about x_A if $x_B = 1$.

A can analyse the transcript and check whether it contains information about x_A. To this end, A determines (by trying all possibilities) for how many values of r'_A with $x'_A = 0$ the transcript T could actually have been generated. and for how many values of r'_A with $x'_A = 1$. If both values are equal (or almost equal), then T does not contain information about x_A, and thus $x_B = 0$. Therefore, A can determine x_B from T.

For the general case $n > 2$ we have two sets $\mathcal{M}_1$ and $\mathcal{M}_2$ such that their union contains all players. One possible function that we could want to compute is that one player of $\mathcal{M}_1$ and one of $\mathcal{M}_2$ have one input bit and want to compute the AND, and no other player has an input. However, if such a program exists, then we could use it to solve the AND problem for $n = 2$. A would simulate all players from $\mathcal{M}_1$ and B the players from $\mathcal{M}_2$.

10.2 The Protocol

We assume that $t < n/2$ and consider an arbitrary function over $GF(q)$ represented as a series of gates for addition and multiplication. In the beginning, the inputs are distributed using a $(t+1, n)$ threshold scheme, and then the computation is performed gate by gate. All inputs, intermediate results and the output are shared using the $(t+1, n)$ threshold scheme.

We have already shown the protocols Share and Reconstruct. What remains is to show how addition and multiplication gates can be evaluated while maintaining the invariant. Addition, and any linear function in general can easily be achieved by just (locally) performing the computation on the shares. This does not require communication, and thus privacy is not an issue, as nobody gains new knowledge.

More difficult is multiplication of two distributed values a and b with result $c = ab$. Every player P_i receives shares a_i and b_i and should have a share c_i for c in the end. The sharing $c_1, \ldots, c_n$ should (from the viewpoint of any set of t players) be completely random, as if an oracle has generated it. We cannot just locally multiply our shares as $d_i = a_i b_i$, as the following problems occur:

- The degree of the resulting polynomial doubles to $2t$. This is still lower than n, but after many multiplications the degree will be larger than n very fast.
- Furthermore, the polynomial is not random (in particular, it is reducible), which would harm privacy.

The key observation is, that the interpolation of any polynomial of degree less than n consists of n polynomial values a linear formula is:

$$f(x) = \sum_{i=1}^{n} \prod_{k \neq i} \frac{x - \alpha_k}{\alpha_i - \alpha_k} s_i$$

Since the values α_i are constant, we can express the secret $s = f(0)$ as weighted sum $s = w_1 s_1 + \cdots + w_n s_n$ of the shares s_i. The product c is therefore

$$c = \sum_{i=1}^{n} w_i d_i \qquad \text{where} \quad w_i = \prod_{k \neq i} \frac{\alpha_k}{\alpha_k - \alpha_i}$$

Thus, the product can be expressed as a linear combination of the values d_i. All one needs to do now is to come up with a way to evaluate the function, but as it is linear, this is easy. Every player shares d_i and applies to (linear) function to his shares. Now, the players have a sharing (of degree t) for c.

11 Broadcast

Against passive parties, broadcast is of course easy to achieve. We request the following properties:

Consistency. All honest parties settle for the same value, that is we have agreement after the protocol.

Validity. If the sender is honest, then all honest parties settle for the same value that the sender sent.

Termination. All honest parties eventually settle for a value.

A similar protocol is *consensus*, where we have the following requirements (every player has an input now):

Consistency. All honest parties settle for the same value $y = y_i$, that is we have agreement after the protocol.

Persistency. If all honest players P_i have the same input value $x = x_i$, then all honest player choose the same output value $y_i = x$.

Termination. All honest parties eventually settle for a value.

For $t < n/2$ we can transform one protocol to the other:

- **Broadcast $\Rightarrow$ consensus**. Every player uses broadcast to tell the other it's own value. The result for every player is the value received most often.

- **Consensus $\Rightarrow$ broadcast**. The sender sends its input to all other players, and then we use consensus.

We will now construct a protocol for broadcast and $t < n/3$ and we will concentrate on a protocol for binary broadcast/consensus.

11.1 Weak Consensus

Every player starts with an input $x_i \in \{0,1\}$ and eventually choose an output value $y_i \in \{0,1,\perp\}$ where we read $\perp$ as "invalid". We request a weaker form of consistency.

Weak Consistency. If one player P_i has output $y_i \in \{0,1\}$ (and not $y_i = \perp$), then for all other honest players P_j we have $y_j \in \{y_i, \perp\}$.

The following protocol, $\texttt{WeakConsensus}(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$, achieves weak consensus.

1. $\forall P_i$: send x_i to every P_j.
2. $\forall P_j$: $y_j = \begin{cases} 0 & \text{if the number of received zeros} \geq n - t \\ 1 & \text{if the number of received ones} \geq n - t \\ \perp & \text{otherwise} \end{cases}$.
3. $\forall P_j$: return y_j.

Lemma 11.1. *The protocol* `WeakConsensus` *achieves persistence, weak consistency and termination.*

Proof. We only show weak consistency via contradiction. Assume there are players P_i and P_j with $y_i = 0$ and $y_j = 1$. At least $n - t$ players sent a zero to P_i, and similar for P_j. Thus, at least $2(n-t) - n$ players have sent inconsistent values to i and j (and these players must thus be dishonest). However, this is more than t. □

11.2 Graded Consensus

Every player starts with input $x_i \in \{0,1\}$ and has outputs $y_i, g_i \in \{0,1\}$, where g_i is a so-called grade value. If $g_i = 0$, then we do not have consistency, otherwise we have. We have the following new requirements.

Graded consistency. If an honest player finishes with y_i and $g_i = 1$, then for all honest players P_i we have $y_i = y_j$.

Graded persistency. If all honest players start with the same value x, then all honest players finish with output $(y_i, g_i) = (x, 1)$.

The following protocol, $\texttt{GradedConsensus}(x_1, \ldots, x_n) \to ((y_1, g_1), \ldots, (y_n, g_n))$, achieves graded consensus.

1. $(z_1, \ldots, z_n) = \texttt{WeakConsensus}(x_1, \ldots, x_n)$
2. $\forall P_i$: send z_i to every P_j

3. $\forall P_j$:

$$y_i = \begin{cases} 0 & \text{if } \#\text{zeros} \geq \#\text{ones} \\ 1 & \text{otherwise} \end{cases}$$

$$g_i = \begin{cases} 1 & \text{if } \#y_j\text{'s} \geq n - t \\ 0 & \text{otherwise} \end{cases}$$

4. $\forall P_j$: return (y_j, g_j).

Lemma 11.2. *The protocol* `GradedConsensus` *achieves graded persistency, graded consistency, and termination*

Proof. Graded persistency. If all honest players have input $x_i = x$, then `WeakConsensus` will guarantee that all players send $z_i = x$. Thus, all honest players receive at least $n - t$ times x, and thus set $y_i = x$ and $g_i = 1$.

Graded consistency. Let players P_i and P_j be honest, and let $g_i = 1$. We have to show that now $y_j = y_i$. From $g_i = 1$ we know that P_i received from $n - t$ players P_k the value $z_k = y_i$. Of these players, at least $n - 2t$ are honest, and have sent the same value also to P_j. From `WeakConsensus` we also know that the values z_k of the honest players do not contradict, i.e., all are in $\{0, \perp\}$ or $\{1, \perp\}$. □

11.3 King Consensus

One (arbitrary) player will take the role of the king. If the king is honest, then we achieve consensus, and otherwise we still want to have preagreement.

King consistency. If the king P_k is honest, then all honest player output the same value $y \in \{0, 1\}$.

The following protocol, $\texttt{KingConsensus}_{P_k}(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$, achieves king consensus.

1. $((z_1, g_1), \ldots, (z_n, g_n)) = \texttt{GradedConsensus}(x_1, \ldots, x_n)$
2. P_k: send z_k to every P_j
3. $\forall P_j \quad y_j = \begin{cases} z_j & \text{if } g_j = 1 \\ z_k & \text{otherwise} \end{cases}$
4. $\forall P_j$: return y_j.

Lemma 11.3. *The protocol* `KingConsensus` *achieves king consistency, persistency, and termination*

Proof. Persistency. If all honest players have input $x_i = x$, then `GradedConsensus` will guarantee that $z_i = x$ and $g_i = 1$, thus all honest players end with $y_i = z_i = x$.

King consistency. Assume P_k is honest. If every player in P_j in step one gets grade value 0, then all honest players accept z_k of the king. Otherwise, if at least one player has $g_j = 1$, then graded consistency says that all honest players P_i receive the same value $z_i = z$, in particular $z_j = z_k$. Thus, every player gets z_k. □

11.4 Consensus and Broadcast

Now we can simply repeat king consensus with $t+1$ different kings, which ensures that at least one honest player was king once.

1. for $k := 1$ to $t+1$ do
$$(x_1, \ldots, x_n) = \texttt{KingConsensus}_{P_k}(x_1, \ldots, x_n)$$
end
2. $\forall P_j$: return $y_j = x_j$.

Finally, we can achieve a protocol, $\texttt{Braodcast}(x) \to (y_1, \ldots, y_n)$, for broadcast.

1. The sender sends x to all P_j
2. $(y_1, \ldots, y_n) = \texttt{Consensus}(x_1, \ldots, x_n)$
3. $\forall P_j$: return y_j.

Theorem 11.1. *The protocols* `Broadcast` *and* `Consensus` *achieve broadcast and consensus, respectively, if $t < n/3$ players are dishonest. The communication and computation complexities are polynomial in n.*

12 Security Against an Active Opponent

We assume a synchronous communication model where messages are guaranteed to arrive after a fixed amount of time. We will look at three types of malicious behaviour:

- **Not keeping secret information secret.** This can be done in one of two forms; (1) the coefficients of secret-sharing are not chosen randomly, or (2) reveal parts (or all) of the secret information to some or all of the other players.
- **Not sending values** that are required to be sent by the protocol.
- **Send invalid values.** This is the most general form of malicious behaviour.

12.1 Not Keeping Secret Information Secret

This is not a problem, as the information of t players is not sufficient to compute intermediate results or the output, even if all these players send the information to others. Note that an honest player could get $t+1$ shares, but as the player is honest, this is no problem. If he would use the information to compute the result, he would be dishonest.

12.2 Not Sending Values

We consider three separate cases:

(1) When sharing a value, a player does not send a share to some of the players.

(2) When sharing a share of a product (in the multiplication protocol), a player does not send a share to some of the players.

(3) During reconstruction of a value, a player does not send its share to some of the players.

Case (3) is easy to handle, as we only need $t+1$ shares to reconstruct a value. If $n \geq 2t+1$, we can tolerate up to t players that do not send a value.

In the other cases, we first make the problem public. If a player does not receive a share, then he tells all the others (using broadcast). Then, the player that did not send a value has to send the share to all players. If he does, then we are fine, as the complaining player now has the share. The fact that the others do as well is ok, because either the dealer or the complaining player are dishonest. On the other hand, if the dealer does not broadcast its share to all players, then the players can see that the dealer is dishonest. For (1), we can now just take some default input for this player (e.g., 0 with 0 as share for everyone). For (2) it is more difficult, as the share is needed for the multiplication protocol. Several solutions exist.

12.3 Sending Wrong Values

12.3.1 A Theoretical Solution

We will attempt to solve the problem of actively cheating opponents (ones that send wrong values) by detecting the invalid values, and interpreting them as "not sending".

- Every player is *committed* to every value he knows.

- When sending a value to another player, we also need to transfer the commitment. This can be achieved by having the sender open the commitment (for the receiver), and having the receiver commit to the value and proving that the committed value is the same as the received value.
- When broadcasting, the commitment has to be opened for every player, and one has to ensure that every player gets the same value (e.g., using consensus).
- For every internal step of the computation of a player, this player has to prove (e.g., using a zero-knowledge proof) that he computed the step correctly. Put another way, he commits himself to the result and proves that the committed value is the same as the correct result.

12.3.2 A Practical Solution

For the commitment scheme presented in Section 12.3.1 we need the following operations:

1. A player can commit to a value.
2. A player that is committed to a certain value can open the commitment for a certain player.
3. The players can transfer commitments that are valid for one player to another one.
4. The players can transform two commitments (for the same player) into a new commitment (for the same player), whose value is the sum of the values from the original commitments.
5. The players can transform two commitments (for the same player) into a new commitment (for the same player), whose value is the product of the values from the original commitments.

The first two are the inherent operations of a commitment scheme; Open and Commit. The forth operation (addition) can be achieved using the *homomorphic* property of a commitment scheme. A commitment scheme is called homomorphic (with regard to addition) if the commitment for the sum of two commitments can be computed locally.

For the third operation we will use a *commitment transfer protocol* (CTP), which can often be achieved trivially by just sending the value and the information to open the commitment. The last operation will use a *commitment multiplication protocol* (CMP).

Lemma 12.1. *Given a homomorphic commitment scheme with CTP and CMP, as well as a broadcast protocol, we can achieve an MPC protocol with the respective type of security and up to $t < n/2$ active opponents (but at most the number of cheaters tolerated by the commitment scheme and broadcast protocol.*

Commitment sharing protocol. We start with a dealer that is committed to some value s and want that in the end every player has a share of s and is committed to it.

1. The dealer chooses the random coefficients used in the secret sharing scheme and commits to them.
2. Each player (locally) computes the commitments to all shares (using the homomorphic property).
3. For every player, the dealer transfers the commitment to the corresponding share to that player using CTP.

Actively secure multi-party computation. Now, the full multi-party communication protocol goes as follows.

Whenever some party refuses to execute any of the steps, we proceed as described earlier.

1. **Input.** Each player give input by committing to it and then executing the CSP to commit all other players to their share of the input.
2. **Computation.** We evaluate the function gate by gate.
   i) **Addition (and linear functions).** Two values are added by having each player commit to the sum of his two shares using the homomorphic property of the commitment scheme (that is, we do not need communication.
   ii) **Multiplication.** To multiply two values, each player first uses the CMP to commit to the product of his shares. Then, he uses the CSP to share his value. Finally, the players compute the product using Lagrange interpolation (as in the passive protocol), which is a linear function an can be evaluated using the homomorphic property.
3. **Output.** To reconstruct a value towards some player, all other players open their commitment to the share of that value to that player. Since there are at least $t+1$ honest players, the player receives enough shares.

13 Information-Theoretically Secure MPC with an Active Opponent

We know that there is no bit commitment scheme that is information-theoretically secure for both players. The trick will be to have a *distributed* scheme. We understand commitments in a more general sense here; a commitment scheme for n players consists of two protocols, Commit and Open.

- The protocol Commit allows the dealer D to commit himself to a value.
- The protocol Open allows D later to open this value again later.

Both protocols can end with the players rejecting the execution (and thus expose D as cheater). We assume $t < n/3$ and have the following requirements:

1. **Consistency.** If Commit or Open are rejected by some honest players, all honest players do so. If D is honest, no honest player rejects.
2. **Uniqueness.** If Commit is successful, then D is committed to some value, i.e., there is only one value that is accepted in Open.
3. **Privacy.** If D is honest and commits to some value, then, in Commit, the adversary obtains no information about this value.

These properties imply that the scheme is perfectly binding and hiding.

13.1 Commitments Using a Polynomial of Degree $\leq t$

The main idea is that the dealer D, to commit to some value s, chooses a random polynomial $g(x)$ of degree at most t with $g(0) = s$ and sends the *commit share* $s_i = g(\alpha_i)$ to P_i. Furthermore, D somehow "proves" that the commit shares indeed lie on a polynomial of degree at most t. The full protocol is:

1. **Distribution.** To share a secret s, the dealer D chooses a random bivariate polynomial at most t

$$f(x,y) = \sum_{i=0}^{t}\sum_{j=0}^{t} f_{ij}x^i y^j, \qquad \text{where} \quad f_{00} = s \text{ and } f_{ij} \in_R GF(q)$$

   and sends the polynomials $h_i(x) := f(x, \alpha_i)$ and $k_i(y) = f(\alpha_i, y)$ to P_i.

2. **Consistency checks.** For $1 \leq i, j \leq n$ the players (P_i, P_j) verify that $h_i(\alpha_j) = k_j(\alpha_i)$. To that end, P_i sends the value $h_i(\alpha_j)$ to P_j, who compares the values. Each player broadcasts a complaint for all coordinates (i, j) where the values do not match. The dealer D must answer such complaints by broadcasting $f(\alpha_i, \alpha_j)$.

3. **Accusations.** If in step 2, the dealer D broadcasts values that are not consistent with the polynomials $h_i(x)$ and $k_i(y)$ for some player P_i, this player accuses the dealer via broadcast. The dealer must answer such an accusation by broadcasting both $h_i(x)$ and $k_i(y)$. If the broadcast polynomials are inconsistent with the polynomials of some other player, this player also accuses the dealer. This continues until no new player accuses D.

4. **Determine commit share.** If the dealer D in step 3 was accused by more than t players, refused to answer any accusations, or the answers contradicted each other, he is disqualified.

   Otherwise, each player computes his share $s_i = k_i(0)$ as the output of the protocol, where, if P_i accused D, he uses the polynomial broadcast by D. The dealer outputs the polynomial $g(x) = f(x, 0)$.

13.2 Opening Distributed Commitments

We can use the following protocol to open a commitment. Suppose the dealer D committed to some value s by some polynomial $g(x)$ of degree at most t, and every player P_i holds $s_i = g(\alpha_i)$. Then:

1. The dealer D broadcasts $g(x)$.
2. Every player P_i checks that his share lies on the broadcast polynomial. If not, he accuses the dealer.
3. If more than t players accused D, then the protocol is rejected. Otherwise the opened value is $s = g'(0)$.

14 Perfectly-Secure MPC with Linear Time Complexity

We consider a MPC scheme with perfect security (i.e., information-theoretic security without error probability) in a setting n players. Up to $t < n/3$ players are actively corrupted by an adversary. The adversary is computationally unbounded.

14.1 Hyper-Invertible Matrices

Definition 14.1 (Hyper-Invertible Matrix)**.** *An $r \times c$ matrix M is called* hyper-invertible *if for any index sets $R \subseteq \{1, \ldots, r\}$ and $C \subseteq \{1, \ldots, c\}$ with $|R| = |C| > 0$, the matrix M_R^C is invertible, where M_R denotes the matrix consisting of the rows $i \in R$ of M, and similar for the columns and M^C.*

Construction of hyper-invertible matrices. Let $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n$ denote fixed distinct elements in a field $\mathcal{F}$, and consider the function $f : \mathcal{F}^n \to \mathcal{F}^n$, mapping $(x_1, \ldots, x_n)$ to $(y_1, \ldots, y_n)$ such that the points $(\beta_1, y_1), \ldots, (\beta_n, y_n)$ lie on the polynomial g of degree $n-1$ defined by points $(\alpha_1, x_1), \ldots, (\alpha_n, x_n)$. Due to the linearity of Lagrange interpolation, f is linear and can be expressed as a matrix $M = \{\lambda_{i,j}\}_{i=1,\ldots,n}^{j=1,\ldots,n}$ where

$$\lambda_{i,j} = \prod_{k \neq j} \frac{\beta_i - \alpha_k}{\alpha_j - \alpha_k}$$

More explicitly, we have

$$y_i = f(\beta_i) = \sum_{i=1}^{n} \left[\prod_{k \neq j} \frac{\beta_i - \alpha_k}{\alpha_j - \alpha_k} \right] x_i$$

That is, we can express the y_i's as a weighted sum over the x_i.

Lemma 14.1. *The construction above yields a hyper-invertible matrix M.*

Lemma 14.2. *Let M be a hyper-invertible $n \times n$ matrix and*

$$(y_1, \ldots, y_n) = M(x_1, \ldots, x_n)$$

Then, for any index sets $A, B \subseteq \{1, \ldots, n\}$ with $|A| + |B| = n$ there exists an invertible linear function $f : \mathcal{F}^n \to \mathcal{F}^n$, mapping the values $\{x_i\}_{i \in A}$, $\{y_i\}_{i \in B}$ onto the values $\{x_i\}_{i \notin A}$, $\{y_i\}_{i \notin B}$.

Part III
Voting

15 Introduction

We consider a setting with N authorities $A_1, \ldots, A_N$ and M voters. We assume to have insecure (but synchronous) channels, and so-called *bulletin boards* (BB). There, every player can read and write, but no-one can delete information. Further, we assume to have a PKI where everyone knows the public key of every one else. We request that at least half of the authorities are honest, and there are no such requirements for the voters.

We have the following security requirements:

- **Correctness.** Only valid ballots must be considered.
- **Privacy.** It is not possible to obtain information on some voter's vote. Moreover, no information on partial results may be leaked (independence).
- **Robustness.** The protocol cannot be aborted.

16 Voting with Homomorphic Encryption

The main ingredient of voting protocols based on homomorphic encryption is a homomorphic public-key encryption scheme. This allows to add two encrypted messages without knowing the secret key. Based on this, one can construct a voting scheme as follows: the authorities jointly create an encryption key pair (in an MPC) and publish the public key while keeping the secret key shared among themselves. A voter can now cast a ballot by encrypting their vote using this public key and placing the ciphertext, along with a signature and a validity proof, on the BB. Finally, the authorities make use of the homomorphism to add up the encrypted votes and decrypt the result.

Let $(G, \oplus)$ be an additive group, and $(H, \otimes)$ a multiplicative group, where we denote subtraction in G by $\ominus$ and division in H by $\oslash$. As we have seen, a function $f : G \to H$ is called homomorphic if for all $x_1, x_2 \in G$ we have

$$f(x_1 \oplus x_2) = f(x_1) \otimes f(x_2)$$

16.1 "OR"-proofs

We want a proof knowledge of a pre-image x of at least one of the values $y_1, \ldots, y_L \in H$. More formally, the prover should show that he knows x and $i \in \{1, \ldots, L\}$ such that $f(x) = y_i$, but without giving any information about i or x away.

In order to achieve this, we will use L parallel instances of the simple pre-image proof, but with only a single challenge c, which can be split by the prover into L sub-challenges. The full protocol is shown in Figure 10.

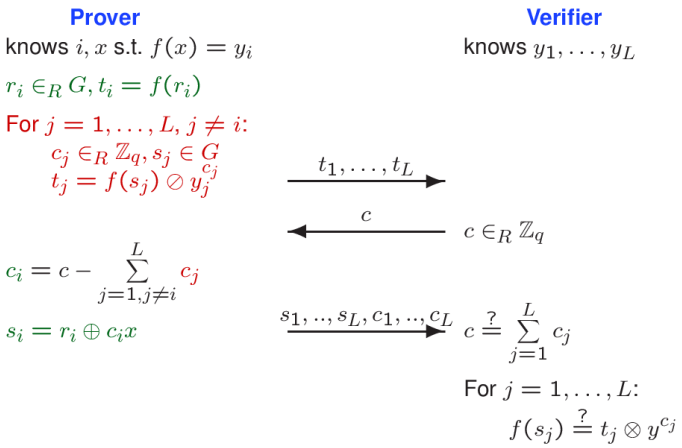

Figure 10: Proving knowledge of a pre-image of several values.

16.2 Homomorphic Encryption

We consider a variant of ElGamal's encryption scheme. Let G be an abelian group of order q, where q is sufficiently large, and let g be a generator of G. Choose secret key $z \in_R \mathbb{Z}_q$ and let $Z = g^z$ be the public key. Encryption is defined by

$$E_Z : \mathbb{Z}_q \times \mathbb{Z}_q \to G \times G, \qquad (v, \alpha) \mapsto (g^\alpha, g^v Z^\alpha)$$

To decrypt a cipher text $e = (x, y)$ one computes (in G)

$$\frac{y}{x^z} = \frac{g^v Z^\alpha}{g^{\alpha z}} = \frac{g^v g^{\alpha z}}{g^{\alpha z}} = g^v$$

and then determines the discrete logarithm to base g thereof (using brute force).

16.3 Distributed Key Generation

To generate a distributed key, we use the PKI already in place. The common private key is $z = \sum_{i=1}^{N} z_i$, and accordingly, the public key $Z = \prod_{i=1}^{N} Z_i = g^z$. We can use the following protocol:

1. Every authority chooses a random $\hat{z}_i \in \mathbb{Z}_q$ and broadcasts the commitment $\hat{Z}_i = g^{\hat{z}_i}$.
2. The secret key z is defined as $z = \sum_{i=1}^{N} \hat{z}_i$, and the public key as $Z = \prod_{i=1}^{N} \hat{Z}_i = g^z$.
   i) Every authority A_i shares its $\hat{z}_i$ using the commitment sharing scheme. A_i chooses a random polynomial $f_i(x)$ with $f_i(0) = \hat{z}_i$ of degree t and commits itself to the coefficients. The share for A_j is $z_{ij} = f_i(\alpha_j)$. The receiver of z_{ij} makes sure that the received value is compatible with Z_{ij}, and otherwise accuses A_i, which then has to broadcast its share.
   ii) Every authority A_j computes its share z_j of z as the sum of the shares received; $z_j = z_{1j} + \cdots + z_{Nj}$. The shares all lie on the polynomial $f(x) = f_1(x) + \cdots + f_N(x)$ and the secret key is $z = f(0)$.

16.4 Distributed Decryption

To decrypt a ciphertext $e = (x, y)$, one must compute x^z:

$$x^z = x^{\sum_{i=1}^{N} z_i} = \prod_{i=1}^{N} x^{z_i}$$

Therefore, if every A_i publishes $X_i = x^{z_i}$ and proves that it was computed correctly, e can be decrypted by computing the discrete logarithm (to base g) of

$$\frac{y}{X_1 \cdot \ldots \cdot X_N}$$

The proof of $X_i = x^{z_i}$ can be done by proving (in zero knowledge) knowledge of a pre-image of (Z_i, X_i) w.r.t. the function

$$f : \mathbb{Z}_q \to G \times G, \qquad \zeta \mapsto (g^\zeta, x^\zeta)$$

It can easily be seen that f is homomorphic, and that $f(0) = (1, 1) = (s, t)^q$ for all $s, t \in G$. Therefore, the theorem from the first part applies with $l = q$ and $u = 0$.

For the computation we need all authorities and thus we have the problem of malicious authorities that refuse to perform some of the steps. To overcome this issue, we can have the authorities share their secret key in a t-out-of-n sharing in the beginning, such that we can recover later if necessary.

16.5 The Protocol

16.5.1 Setup

In the setup phase of the protocol, the authorities generate the public and secret key. Also, the agree on the "topic" of the election, as well as the set $\mathcal{V}$ of valid votes.

16.5.2 Casting a Vote

In order to cast a vote $v \in \mathcal{V} \subseteq \mathbb{Z}_q$, the voter encrypts v as $e = E(v, \alpha)$ for some randomness $\alpha \in_R \mathbb{Z}_q$. Then, they construct a validity proof (as described below), sign both the vote and the proof, and post the entire package to the BB.

The validity proof is an "OR"-proof of many instances of a protocol that proves the statement for a particular vote $v_i \in \mathcal{V}$.

Consider $e = E(v_i, \alpha)$ and the function

$$f : \mathbb{Z}_q \to G \times G, \qquad \alpha \mapsto E(0, \alpha)$$

Clearly, f is homomorphic. Moreover, $f(0) = (1, 1) = (s, t)^q$ for all $s, t \in G$. Thus we can use the theorem of the first part. The voter now proves knowledge of a pre-image of $e \oslash E(v_i, 0)$ w.r.t. f. The existence of such a pre-image implies that e contains v_i.

Finally, when combining the above proofs in an "OR"-proof, the voter uses the Fiat-Shamir heuristic to make the proof non-interactive.

16.5.3 Tallying

The authorities can compute an encryption e_T of the result T by summing up all (valid) votes. This ciphertext is then decrypted to obtain T.

17 Receipt Freeness

In order to prevent vote-buying and coercion, we want a protocol that does not allow a voter to prove which way he voted. Such a protocol is called *receipt-free*.

17.1 Model

In general, we cannot achieve this, as a vote buyer can eavesdrop on the communication between the voter and the authorities. The vote buyer can then use the same inputs and run an interactive turing machine (ITM) that specifies the behaviour of an honest voter. This input to the ITM is a receipt, as the vote buyer can just check whether the transcript generated is the same

Therefore, we use a stronger model where we have (physically) secure channels between a special authority (the randomizer) and the voters. Furthermore, none of the authorities communicate with the vote buyer.

17.2 Receipt-Freeness Using a Randomizer

Again the voting scheme is based on homomorphic encryption, and on a high level works as follows. The voter sends its encrypted vote e to the randomizer, who randomizes e by adding an encryption of 0, resulting in e^*, and proves to the voter that he executed this step correctly. Moreover, with the voter's help, he generates a validity proof of the vote. Finally, the voter signs the randomized ciphertext and everything is sent to the BB.

17.2.1 Randomization Proof

To prove to the voter that e^* is a correct randomization, the randomizer could prove knowledge of a pre-image α of $e^* \oslash e$ with regard to the homomorphism

$$f : \mathbb{Z}_q \to G \times G, \qquad \alpha \mapsto E(0, \alpha)$$

as this would imply that e and e^* contain the same value. However, then the voter could just forward this proof.

We combine this proof with a proof that the randomizer knows the secret key of the voter in an "OR"-proof. This is a *designated-verifier proof*, as it only convinces the voter.

17.2.2 Validity Proof

Finally, we show how the voter and randomizer can jointly construct a non-interactive proof of validity for e^*. The main idea is that the voter and the randomizer execute the validity proof for e, and at the same time, the randomizer randomizes the proof and transforms it into a non-interactive proof of validity for e^*.

Let $L = |\mathcal{V}|$. Recall the "OR" proof of Figure 10. The randomizer now randomizes this proof by choosing

$$r'_1, \ldots, r'_L, c'_1, \ldots, c'_L \in_R \mathbb{Z}_q \quad \text{such that} \quad \sum_{i=1}^{L} c'_i = 0$$

and setting, for $j = 1, \ldots, L$,

$$r''_j = r_j + r'_j, c''_j = c_j + c'_j \quad \text{and} \quad t''_j = t_j \otimes E(c'_j v_j, r'_j) \oslash e^{c'_j}$$

The tuple $(t''_1, \ldots, t''_L, c, r''_1, \ldots, r''_L)$ is an accepting conversation. Moreover, it is independent of the original transcript. It remains to adapt the proof such that the transcript is accepting for proving that $e^* = e \otimes E(0, \alpha)$ is a valid vote. That is, for $j = 1, \ldots, L$, the relation

$$E(0, r_j) \stackrel{?}{=} t_j \otimes (e \otimes E(0, \alpha) \oslash E(v_j, 0))^{c_j}$$

must be satisfied. This can be achieved by setting

$$r''_j = r_j + r'_j + \alpha c''_j$$

The protocol to generate the validity proof between the randomizer and the voter now works as follows. The voter executes the normal validity proof, taking the role of the prover. The randomizer, however, given the first messages $t_1, \ldots, t_L$ by the voter, computes $t''_1, \ldots, t''_L$ as described above and applies a hash function to them in order to obtain the challenge c, which he sends to the voter. From the voter's response, he can compute $c''_1, \ldots, c''_L$ and $r''_1, \ldots, r''_L$ as above and the tuple

$$(t''_1, \ldots, t''_L, c''_1, \ldots, c''_L, r''_1, \ldots, r''_L)$$

is the non-interactive proof of validity of e^*. Finally, the randomizer puts e^* along with the proof above onto the BB. The voter places a signature for e^* onto the BB in order to accept e as his ballot.

References

[Mau09] Ueli Maurer. Unifying zero-knowledge proofs of knowledge. In *Proceedings of the 2nd International Conference on Cryptology in Africa: Progress in Cryptology*, AFRICACRYPT '09, pages 272–286, Berlin, Heidelberg, 2009. Springer-Verlag.