Efficient Generic Zero-Knowledge Proofs from Commitments

Samuel Ranellucci * **, Alain Tapp, and Rasmus Zakarias* **

[1] Department of Computer Science, Aarhus University
{samuel,rwl}@cs.au.dk
[2] DIRO, Université de Montréal, Canada,
tappa@iro.umontreal.ca

Abstract. Even though Zero-knowledge has existed for more than 30 years, few generic constructions for Zero-knowledge exist. In this paper we present a new kind of commitment scheme on which we build a novel and efficient Zero-knowledge protocol for circuit satisfiability. We can prove knowledge of the AES-key which map a particular plaintext to a particular ciphertext in less than 4 seconds with a soundness error of 2^{-40}. Our protocol only requires a number of commitments proportional to the security parameter with a small constant (roughly 5).

1 Introduction

Zero-knowledge was introduced in 1985 by Goldwasser, Micali and Rackoff in their seminal paper [12] introducing the IP hierarchy for interactive proof systems and the concept of Zero-knowledge complexity.

Informally, a Zero-knowledge argument is an interactive protocol that allows a Prover to persuade a Verifier of the validity of some NP statement using knowledge of some hidden witness. Essentially, the Verifier should learn nothing more than the fact that the Prover knows a witness that satisfies the statement.

One motivating example is that of graph isomorphism: the NP statement here is that two graphs are isomorphic. The witness is a permutation held by the Prover permuting one graph into the other. One obvious way for the Prover to convince the Verifier would be to send the permutation. However, this reveals much more information than the one bit of information to be conveyed, namely whether the graphs are isomorphic or not. Zero-Knowledge proofs are interactive proof systems ensuring that the Verifier learns only this information and nothing more.

* The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within part of this work was performed; and from the CFEM research center, supported by the Danish Strategic Research Council.
** Supported by the European Research Council Stating Grant 279447

Following this ground breaking work, [1] showed that for any relation that can be proven by an interactive proof systems, it can also be proven in Zero-knowledge. Thus, the potential for applications of Zero-knowledge are expansive. A large body of work has shown that specialized efficient constructions for specific NP relations are possible. However, even though Zero-knowledge has existed for almost 30 years, generic constructions for Zero-knowledge are very few. Moreover, the generic constructions that do exist, use the relatively impractical Karp reductions [15] to NP-complete languages [11].

Generic constructions for Zero-knowledge are starting to emerge. The recent line of work starting with [14] focus on the novel idea of using garbled circuits for Zero-knowledge proofs of generic statements. This line of work was continued by Frederiksen et al. in [9] where they build specialized garbling schemes tailored for Zero-knowledge proofs. The garbling approach communicates at least one symmetric encryption per And-gatein the circuit. Similary, for security parameter κ and circuit size n our protocol exhibit worst case complexity $O(n\kappa)$. In contrast to the garbling approach, our protocol only requires a small constant of bits by And gate (roughly 5). On the other hand, we have [13] with worst case complexity $O(n)$ when the Prover uses the Scalable Multiparty computation technique from [8]. However this construction is quite involved and even 9 years after its publication no implementation is provided yet[3]. Naturally for large enough circuits [13] will be faster than our construction. Yet we boldly conjecture that our scheme, given its smaller constants, out performs their (in terms of execution time) construction for practical application sized circuits like AES ($\sim 40K$ gates).

Another main selling point for our construction is that it is conceptually simple: only an understanding of the notion for commitments and XOR-sharing are necessary to master it. Also, we demonstrate that our construction is practical by presenting an implementation which exhibit small running times. In particular we present an example application where the Prover proofs knowledge of an AES-encryption-key encrypting a particular public plain-text to a particular public cipher-text. To sum up, our protocol is no novelty in terms of asymptotic complexity, however we emphasize that the concrete constants are small, its construction is conceptually simple (an advantage when selling it to non-crypto experts) and it takes only a moderate effort to implement.

2 Contributions

In this paper, we present a novel approach for achieving generic Zero-knowledge proofs. Similar to the line of work using garbled circuits, our construction uses the idea of proving knowledge of a satisfying assignment for a circuit. However our construction differs from the garbling approach in several ways. Our construction is very simple and clean using only one primitive, a commitment scheme. Our construction is similar to [17] which over a field $\mathbb{F}$ has complexity $O(|\mathbb{F}|n)$ for a

[3] Though the authors say an implementation is under way.

field of size $|\mathbb{F}| > \kappa$. Their approach operates on a gate by gate basis. As our field is small $\mathbb{Z}_2$ we bundle the entire circuit together obtaining a string of size $O(n)$ and prove all gates in one go with soundness $\frac{1}{4}$ and then we repeat 2κ times.

For a simple construction our scheme can be instantiated in the Random Oracle Model which is also rather efficient. However this introduces the extra assumption of the Random Oracle. Recent work on commitments in the standard model proves to be even more efficient and adds no additional assumptions, see the PKC 2015 paper in [4] (improving on [7]). Using their scheme makes our construction gets extremely efficient, since it only relies on encoding for a linear code like Reed Solomon which can be done efficiently using the FFT-transform.

In a bit more detail, we take a similar approached to the one used in [2,6]. They use Xor operations over individually committed bits to prove statements. We employ strings in our construction, using a novel way of committing to bit-strings that enables Zero-knowledge proofs of linear relations. In particular we present efficient protocols for proving equality and inequality of bits in a string given two regular commitments to the Xor sharing of that string. From this we build protocols for circuit satisfiability where a Prover proves to a Verifier that he has knowledge of a witness w that satisfies the circuit. The Prover does this by committing to a truth assignment of all gates in the circuit along with some additional information. Then he proves relationships (corresponding to the gates of the circuit) between bits in the committed string.

By the hiding property of the commitment the Verifier learns nothing about the inputs to the circuit. In the end the Prover essentially opens the output bit of the circuit by proving that the output bit of the circuit committed to is one and this is essentially the only new information that the Verifier learns.

For a circuit of size n with ι input gates, α and gates and β linear gates our construction communicates $4\alpha + \beta$ bits of data with soundness one-quarter. To form a secure protocol with security $2^{-\kappa}$, we repeat our construction 2κ times realizing a protocol with communication complexity $O(\kappa n)$. We emphasize once again that the constants involved are small.

3 Commitment with Linear proofs

In this section, we define a commitment scheme which allows a Prover to prove linear relationships between bit-positions within a string he has committed to. These relationships include equality and inequality. From the (in)equality proofs we build a protocol for proving that a set of bit positions xor to a particular value. The proofs are complete and Honest-Verifier Zero-knowledge. The soundness only holds with probability one-half. We will use the notation $\texttt{xom}(m)$ to say that a Prover commits to a message m allowing him to conduct proofs of linear relations between individual bit positions in m.

The proofs will take the form of a sigma protocol: Three messages are communicated, where the Verifier's challenge will consist of one bit. Proofs within a single string can be combined similar to how sigma protocols can be combined. If a commitment would take part in two tests which have distinct challenges,

then the committed value is revealed. This implies that the soundness of the proofs cannot be improved.

In the following we are going to work on strings, therefore we need a bit of notation. Then follows our xor-commitment scheme.

Notation For an l-bit string m we denote m_i the i'th bit of m. When a message m is xor-shared, we denote m^0 and m^1 the xor-shares of $m = m^0 \oplus m^1$. We sometimes combine these notations and take m_i^0 to mean the i'th bit of the 0'th share of m and similarly for m_i^1. Informally, we say two distributions U and V are *indistinguishable* if for any x sampled from either U or V at random and for all probabilistic polynomial time Turing machines A, running A on x making it guess on which distribution x belongs to is correct with probability $\frac{1}{2}$. For formal formulation and a detailed treatment of indistinguishability we refer the reader to [5].

Commitment scheme supporting linear proofs We will now give the details of our commitment scheme supporting linear proofs. Our scheme relies on any classical commitment scheme from the literature with the desired *Hiding* and *Binding* properties, we denote it $\texttt{com}()$. In following we describe how we *commit* to messages and *open* our commitments.

To commit to a string $m \in \{0,1\}^l$ we first choose a string $r \in \{0,1\}^l$ uniformly at random and set $m^0 = r$ and $m^1 = m \oplus r$. Then we commit to both strings using the classical commitment scheme to form $\texttt{xom}(m) = (\texttt{com}(m^0), \texttt{com}(m^1))$.

To open an $\texttt{xom}(m)$ entirely we open both classical commitments and reveal m. The commitment protocol is depicted in Figure 1.

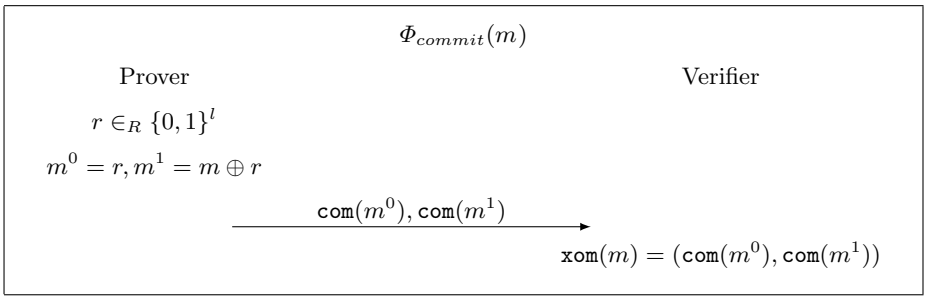


Fig. 1. Commit

3.1 Partial opening

In cases where we wish to do linear proofs, we will open the commitment partially meaning that we only open one of the standard commitments associated to $\texttt{xom}(m)$. We denote $\texttt{Reveal}(M, t)$ the action of a Prover opening the value of the standard commitment $\texttt{com}(m^t)$.

3.2 Properties with xom(·)

We give here a set of theorems stipulating important but strait forward facts regarding XOR shared strings.

Theorem 1. *For a given string $m \in \{0,1\}^l$ and $k \in \{0,1\}$ let V_k be the distribution on strings*

$$\left\{m^k \mid m = m^0 \oplus m^1, m^0 = r, r \in_R \{0,1\}^l\right\}$$

V_k is indistinguishable from the uniform distribution U on $\{0,1\}^l$.

Proof. The proof follows trivially from r being chosen uniformly at random. □

Definition 1. *We say a bit string $m' \in \{0,1\}^l$ looks* independent *of another string m if the distribution from which m' was drawn is indistinguishable from the uniform distribution U on $\{0,1\}^l$.*

Theorem 2. *If two bit position m_i and m_j are equal for a string $m = m^0 \oplus m^1$ then exists δ such that $\delta = m_i^0 \oplus m_j^0 = m_i^1 \oplus m_j^1$.*

Proof.

$$0 = m_i \oplus m_j = m_i^0 \oplus m_j^0 \oplus m_i^1 \oplus m_j^1 \Rightarrow m_i^0 \oplus m_j^0 = m_i^1 \oplus m_j^1 \stackrel{def}{=} \delta.$$

□

Theorem 3. *For two bit positions m_i and m_j the view (δ, m^k) looks independent of m as long as m^{1-k} stays hidden.*

Proof. By Theorem 1. m^k is independent of m and $\delta = m_i^k \oplus m_j^k$, thus no new information is revealed. □

Theorem 4. *For two bit positions m_i and m_j, if $m_i \neq m_j$ then there exists ϵ such that $\epsilon = m_i^0 \oplus m_j^0 = 1 \oplus m_i^1 \oplus m_j^1$.*

Proof.

$$1 = m_i \oplus m_j = m_i^0 \oplus m_j^0 \oplus m_i^1 \oplus m_j^1 \Rightarrow m_i^0 \oplus m_j^0 = 1 \oplus m_i^1 \oplus m_j^1 \stackrel{def}{=} \epsilon.$$

□

Theorem 5. *For two bit positions m_i and m_j the view (ϵ, m^k) is independent if m as long as m^{1-k} stays hidden.*

Proof. Again by Theorem 1 m^k is independent of m. If $k = 0$ then $\epsilon = m_i^0 \oplus m_j^0$ in which case ϵ can be perfectly simulated from what is already revealed. Similarly if $k = 1$ then, $\epsilon = 1 \oplus m_i^1 \oplus m_j^1$ which can be perfectly simulated too. □

In the following we will use the Theorems above to generate proofs for linear relationships between independent positions in a string m that a Prover has committed to.

4 Zero Knowledge with weak soundness

In our first result we show how to do an honest Verifier zero knowledge equality proof between two bit positions of a string m given $\texttt{xom}(m)$. The basic idea is as follows: we will exploit Theorem 2 saying that if the bits m_i and m_j are equal then there exists a δ such that $m_i^0 \oplus m_j^0 = m_i^1 \oplus m_j^1 = \delta$. On the other hand, if $m_i \neq m_j$ by Theorem 4 then no such δ exists. The protocol is depicted in Figure 2.

Informally, observe that by Theorem 3 the view (δ, m_i^b, m_j^b) looks independent of the value of m_i and m_j as long as (m_i^{1-b}, m_j^{1-b}) stay hidden. For *completeness* we observe that if the statement is true e.g. $m_i = m_j$ then $\delta = m_i^b \oplus m_j^b, b \in \{0,1\}$ and the Verifier accepts. For *soundness* consider the first step of the protocol: the Prover will reveal δ. This reveals no information and forces a cheating Prover to prepare to answer a b' such that $m_i^{b'} \oplus m_j^{b'} \neq \delta$. Then the Verifier will select a b at random for which the Prover can only reply correctly if $b = b'$. This ensures that a cheater gets caught with probability one-half. The soundness is evident since any challenge can be easily answered. It is Zero-knowledge since the values of (δ, m_i^b, m_j^b) are independent of the bits m_i and m_j.

4.1 Protocol for equality

In this Section we formally prove the protocol in Figure 2 Honest Verifier zero knowledge with soundness $\frac{1}{2}$. For a string $m \in \{0,1\}^l$ let $M = \texttt{xom}(m) = (\texttt{com}(m_0), \texttt{com}(m_1))$, we show how to prove for a given i, j that $m_i = m_j$.

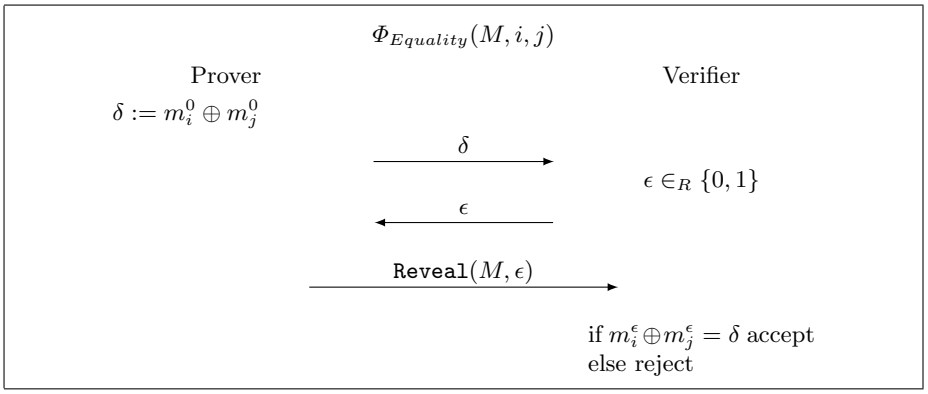


Fig. 2. Protocol for proof of Equality given an $\texttt{xom}(m)$ to a message m. More precisely, the $M = \{m^0, m^1, (\texttt{com}(m^0), \texttt{com}(m^1))\}$ given as input is taken to mean that the Prover already committed to a message m and the Verifier received the commitment.

Theorem 6. *The protocol in Figure 2 is Honest Verifier Zero Knowledge with soundness* $\frac{1}{2}$.

Proof. We show *completeness*, *soundness* and *Honest Verifier Zero knowledge*.
Completeness:
To show completeness, we show that an honest Verifier will be convinced by an honest Prover. Therefore, assuming $m_i = m_j$, we consider two cases:
case 1: if $m_i^0 = m_j^0$ then $m_i^1 = m_j^1$ in which case $\delta = 0 = m_i^0 \oplus m_j^0 = m_i^1 \oplus m_j^1$ and thus the check the Verifier make is true for both choices of ϵ and thus he accepts.
case 2: if on the other hand $m_i^0 \neq m_j^0$ then $m_i^1 \neq m_j^1$ in which case $\delta = 1 = m_i^0 \oplus m_j^0 = m_i^1 \oplus m_j^1$ and the Verifier accepts. These cases are exhaustive assuming that $m_i = m_j$.
Soundness:
To show soundness holds with probability one-half, we consider a cheating Prover and show that an honest Verifier accepts with probability at most one-half. That is, lets assume $m_i \neq m_j$ and consider a Prover try to convince the Verifier otherwise. If $m_i^0 \neq m_j^0$ then $m_i^0 \oplus m_j^0 \neq m_i^1 \oplus m_j^1$ and therefore for any $\delta \in \{0,1\}$ there exists a value ϵ such that the Verifier will not accept by Theorem 4. Therefore, such a cheater is detected with probability $1/2$.
Honest Verifier Zero-Knowledge:
To prove Zero Knowledge for a Verifier, we give a simulator that generates the view $(\texttt{com}(m^0, \texttt{com}()1), \delta, m^b)$ which is indistinguishable from a real execution.

Simulator $M_{V^*}^{Eq}$

1 Pick $m, r \in_R \{0,1\}^l$ s.t. $m_i = m_j$ and set $m^0 = r, m^1 = m \oplus m^0$.
2 Compute $\delta \leftarrow m_i^0 \oplus m_j^0$ and output $C = (\texttt{com}(m^0), \texttt{com}(m^1), \delta)$
3 Accept to receive the $\epsilon \in \{0,1\}$
4 Open the commitment $\texttt{com}(m^\epsilon)$.

Fig. 3. Simulator for equality

Consider the simulator in Figure 3. We argue that the view generated is indistinguishable from a real execution by noticing that it performs the exact same steps as an honest Prover does. From this, we conclude that the view $(\texttt{com}(m^0), \texttt{com}(m^1), \delta, m^b)$ is indistinguishable from a real execution. □

It follows that the above Zero-knowledge proof is for a quite trivial statement, namely that there exists a string m such that $m_i = m_j$. We will use this primitive as a building block in more complicated proofs as we shall see shortly.

4.2 Parallel equality proofs

The scheme above allows the Prover to prove equality for multiple positions $\{(i_k, j_k)\}_k$, however keep in mind that the challenge ϵ used has to be the same otherwise the bits m_i and m_j are revealed. For an xor-commitment $M = \texttt{xom}(m)$ and for a set of pairs of indices $\{(i_v, j_v)\}_{v=1,\ldots,t}$ into m we can prove all pairs

of positions equal with soundness one-half. We summarise the construction for parallel equality in Figure 4.

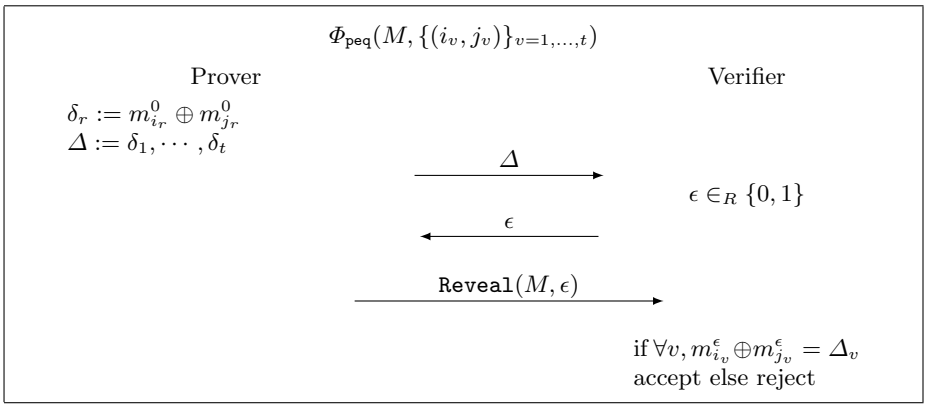


Fig. 4. Parallel Equality

As before we take $M = \{m^0, m^1, (\mathtt{com}(m^0), \mathtt{com}(m^1))\}$ as input to mean an XOR commitment already happened beforehand. For the paired index set $I = \{(i_r, j_r)\}_{r=1,\ldots,t}$ we generate the δ_r and send a string $\Delta = \delta_1, ..., \delta_t$ rather than one bit, to the Verifier. Also, the Verifier now checks t positions one for each bit in Δ.

Following the same reasoning as for the equality protocol in Figure 2 our protocol for parallel equality in Figure 4 is also Honest Verifier Zero Knowledge with soundness one-half. The proof is omitted here as a later proof will show something even stronger.

4.3 Proof of inequality

The proof of inequality is very similar to the equality proof. The main difference is that there exists a ϵ such that for any $b \in \{0, 1\}$, $\epsilon = m^b_i \oplus m^b_j \oplus b$ instead if the $\delta = m^0_i \oplus m^0_j$ as before. We put the protocol and its proof in Appendix C.

4.4 The Linear Zero Knowledge proof

In the previous sections we have seen how to do equality and inequality proofs of bits in a committed message. We can combine these two in one protocol to convince our Verifier that a set of bit positions XOR to a particular value, which covers equality and inequality as special cases.

Now motivated we describe a protocol for convincing a Verifier that a set of bit-position will XOR to a particular value. As before the Prover commits to a message m producing $M = \{m^0, m^1, (\mathtt{com}(m^0), \mathtt{com}(m^1))\}$. Additionally the input of the protocol will be two things. The first thing will be a set of indices,

I, and the second will be the expected value, b, that the bit positions $m_i, i \in I$ are supposed to XOR to. More precisely, our protocol shall prove:

$$(\bigoplus_{i \in I} m_i) = b$$

This captures our previous protocols for (in)equality. E.g. we see that equality is covered by putting two elements in the set and setting the expected value to 0. By expecting a one instead we have inequality. In addition, by only putting a single index in the set, it is a proof that m_i is equal to the given expected value without revealing anything else. Essentially we can open a single bit position in this way. Figure 5 below depicts our protocol for proving the XOR relation between bit positions.

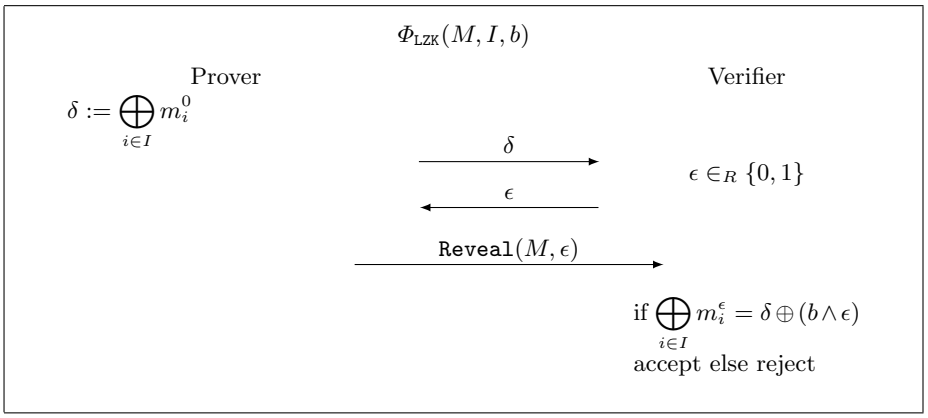


Fig. 5. Linear Zero-knowledge

Theorem 7. *The Linear Zero-knowledge protocol in Figure 5 is Honest Verifier Zero-knowledge.*

Proof. We show *Completeness*, *Soundness error one half* and *Honest verifier zero knowledge*.

Completeness Assuming honest parties the Prover sends δ created as stipulated while also the claim to be proven holds. We consider four cases based on whether we are proving even ($b = 0$ in analogues to equality) or odd ($b = 1$ in analogues to inequality) parity with challenge $\epsilon \in \{0, 1\}$. In more detail we consider ($b = 0, \epsilon = 0$), ($b = 0, \epsilon = 1$), ($b = 1, \epsilon = 0$) and ($b = 1, \epsilon = 1$). We collapse the two first cases to one case $b = 0$.

$b = 0$**:**

$$0 = (\bigoplus_{i \in I} m_i) = \bigoplus_{i \in I} (m_i^0 \oplus m_i^1)$$
$$\Rightarrow$$
$$\bigoplus_{i \in I} m_i^0 = \bigoplus_{i \in I} m_i^1 = \delta$$

Because $b = 0$ our Verifier checks that

$$\delta = \bigoplus_{i \in I} m_i^\epsilon = \bigoplus_{i \in I} m_i^0 = \bigoplus_{i \in I} m_i^1$$

Thus the Verifier accepts.

$b = 1$**:**

We are proving odd parity in this case, that is the bit positions $m_i, i \in I$ has an odd number of one bits. Thus there is no common δ between the XOR of m_i^0 and $m_i^1, i \in I$ in similarity to the inequality we have:

$$1 = (\bigoplus_{i \in I} m_i) = \bigoplus_{i \in I} (m_i^0 \oplus m_i^1)$$
$$\Rightarrow$$
$$\bigoplus_{i \in I} m_i^0 = 1 \oplus \bigoplus_{i \in I} m_i^1 = \delta$$

That is whether $\delta = 1 \oplus \bigoplus_{i \in I} m_i^\epsilon$ is now dependent on ϵ and the fact that we are proving odd parity, $b = 1$. Hence our Verifier now checks:

$$\delta = (\bigoplus_{i \in I} m_i^\epsilon) \oplus (b \wedge \epsilon)$$

And accepts.

Soundness Assuming the claim is false, $\bigoplus_{i \in I} m_i \neq b$, our cheating Prover will attempt to convince the Verifier otherwise. We will show such a Prover succeeds with probability one half.

Sending the $\delta = \bigoplus_{i \in I} m_i^0 \oplus \epsilon'$ to the Verifier the Prover can reply correctly only for one $\epsilon' \in \{0, 1\}$, namely if

$$\delta = (b \wedge \epsilon') \oplus \bigoplus_{i \in I} m_i^{\epsilon'}.$$

Thus with probability one half the Verifier sends $\epsilon \neq \epsilon'$ in which case sending m^ϵ (to satisfy the commitment in M) the Verifier sees an inconsistency with δ and aborts. Thus by case analysis we have established that the cheating Prover is caught with probability one half.

Honest Verifier Zero-knowledge We give here a simulator producing the exact same distribution as in the protocol in Figure 6. This simulator performs the exact steps of the Honest Prover expect choosing the underlying message at random. By the *hiding* property of the underlying commitment the view $(\delta, \texttt{com}(m^0), \texttt{com}(m^1), m^\epsilon)$ exhibit the same flavor of indistinguishability as the hiding property for the underlying commitment scheme. E.g. if the commitment

Simulator $M_{V^*}^{LZK}$

1 Pick at random $m_i, r \in \{0,1\}^l, i \in I$.
2 Compute $\delta = \bigoplus_{i \in I} m_i^0$ and output $(\texttt{com}(m^0), \texttt{com}(m^1), \delta)$ to the Verifier V^*.
3 Accept $\epsilon \; in \{0,1\}$.
4 Open $\texttt{com}(m^\epsilon)$

Fig. 6. Simulator for Linear Zero Knowledge

is computationally hiding then the output distribution of our simulator is computationally indistinguishable from the uniform distribution on the set of strings of the same length. □

Motivated by our parallel proofs for (in)equality in the previous section we give a protocol for packing multiple Linear Zero Knowledge proofs (LZK's) together at no extra cost still using only one commitment. It is depicting in Figure 7 which in more details is a protocol showing how to pack multiple LZK proofs together into one protocol still using only one $\texttt{xom}(\cdot)$. Formally, we pack t instances of the LZK protocol into one protocol taking a set of pairs of indices and expected values. As with parallel equality we now send a string $\Delta = \delta_1, ..., \delta_t$ rather than a single bit in the first step of the protocol. Also, the Verifier checks t sets of indices against t expected values.

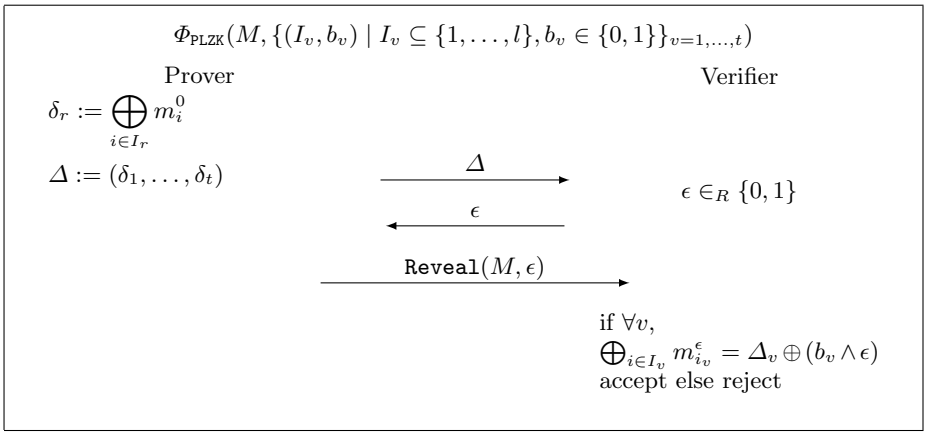


Fig. 7. Parallel Linear Zero-knowledge

Theorem 8. *The protocol in Figure 7 is Honest Verifier zero knowledge with soundness error* $\frac{1}{2}$

Proof. We prove *completeness*, *soundness* and *Honest Verifier Zero Knowledge*.

Completeness We show the Verifier accepts if both Prover and Verifier are honest. By Theorem 8 the Verifier accepts for each δ_i. Thus he accepts for Δ

Soundness The Prover cheats in at least one bit position and by Theorem 8 he get caught with probability one half. As bit positions are independent the Prover only decreases his success probability by cheating in multiple positions. Thus we conclude we have soundness one error probability one half.

Honest Verifier Zero Knowledge We give a simulator producing the values the Verifier sees distributed perfectly as in our protocol in Figure 7. □

Simulator $M_{V^*}^{PLZK}$

1 Sample $m, r \in_R \{0,1\}^l$ with the restriction that for $v = 1, ..., t : \oplus_{i \in I_v} m_i = b_v$ let $m^0 = r, m^1 = m \oplus m^{0\,a}$.
2 Compute $\Delta = \delta_1, ..., \delta_t, \delta_v = \oplus_{i \in I_v} m_i^0$.
3 Output $(\mathtt{com}(m^0), \mathtt{com}(m^1), \Delta)$.
4 Accept to receive the $\epsilon \in \{0,1\}$
5 Open commitment $\mathtt{com}(m^\epsilon)$.

[a] we assume the restrictions in I_v are satisfiable with no contradicting assignments to bit positions in m.

Fig. 8. Simulator for Linear Zero Knowledge

5 And-proof

In this section, we wish to be able to prove that for an $\mathtt{xom}(m)$ to a string m and for three indices (i, j, k) of the string, it holds that $m_i \wedge m_j = m_k$. This will be done by using a helper triple of values that will be made explicitly for this purpose.

To construct such a proof, we will exploit the following relationship:

$$x \wedge y = z \text{ if and only if } z = Maj(x, y, 0).$$

We will have an additional three bits per And-gate which will be a random permutation of the two input values and zero. The protocol for the proof is depicted in Figure 10 and proceeds as follows: the Verifier will for each And-gate, randomly ask that the Prover to either show that the three committed bits are a permutation of the two inputs and zero or show that the majority of the three additional bits is the value of the output value (e.i. an And-gate cf. the relationship above). If the bits do not form a valid triple then the Prover will fail one of two tests. The equality tests that are involved in either Permutation test or majority test above can be cheated with probability one-half. Thus in overall a cheating Prover will get caught with probability one-quarter.

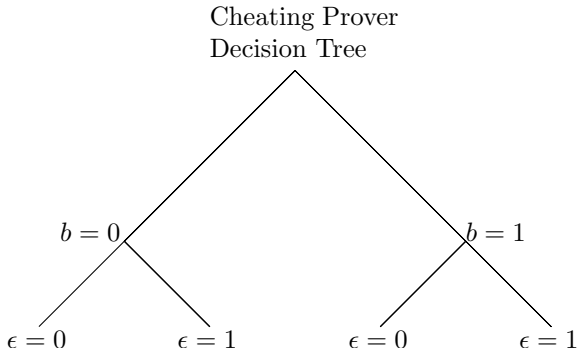


Fig. 9. Overview of the choices a cheating Prover will have in the And-protocol in Figure 10: he can prepare to respond correctly for one choice of b but not the other. Also, for the equality proofs he can prepare to respond correctly for one of values in the ϵ challenge. Thus in three of four cases the cheating Prover wins.

The proof that this protocol is Honest Verifier Zero-knowledge follows from a later proof where we where we combine all our constructs to prove circuit satisfiability. First we will describe a circuit notation before getting to that.

5.1 Circuit notation

In this section, we define a circuit notation which is convenient for our zero-knowledge proofs. We consider circuit over the AND, XOR basis. Indices will be used to identify wires, we assume the description C of a circuit includes a public index set I enumerating the wires. When there is a fork in the circuit, the input of the fork and the outputs of the fork will share the same index. Each pair of wires which are not forked will have different indices. The Xor gates will be represented as a triple of values where the first two values will be the index of the input wires and the third will be the index of the output wire. An And gate will consist of 3 indices, the first and second values will be the input wires and the third value will be the index of the output wire.

We denote l the number of wires and α as the number of and gates.

5.2 The Protocol for circuit satisfiability

Our main result and protocol for circuit satisfiability is given in Figure 12 and we give proof of that the protocol is Complete, Sound and Honest Verifier Zero-knowledge in this Section. In the following Section 6 we show this protocol when repeated multiple times realize the Universally Composable Secure. Thus the observant reader may find that the protocol presented here could be simpler while still achieving Honest Verifier Zero-knowledge.

We assume a public circuit known to both Verifier and Prover. In addition the Prover knows a witness w satisfying the circuit. The circuit consists of AND and

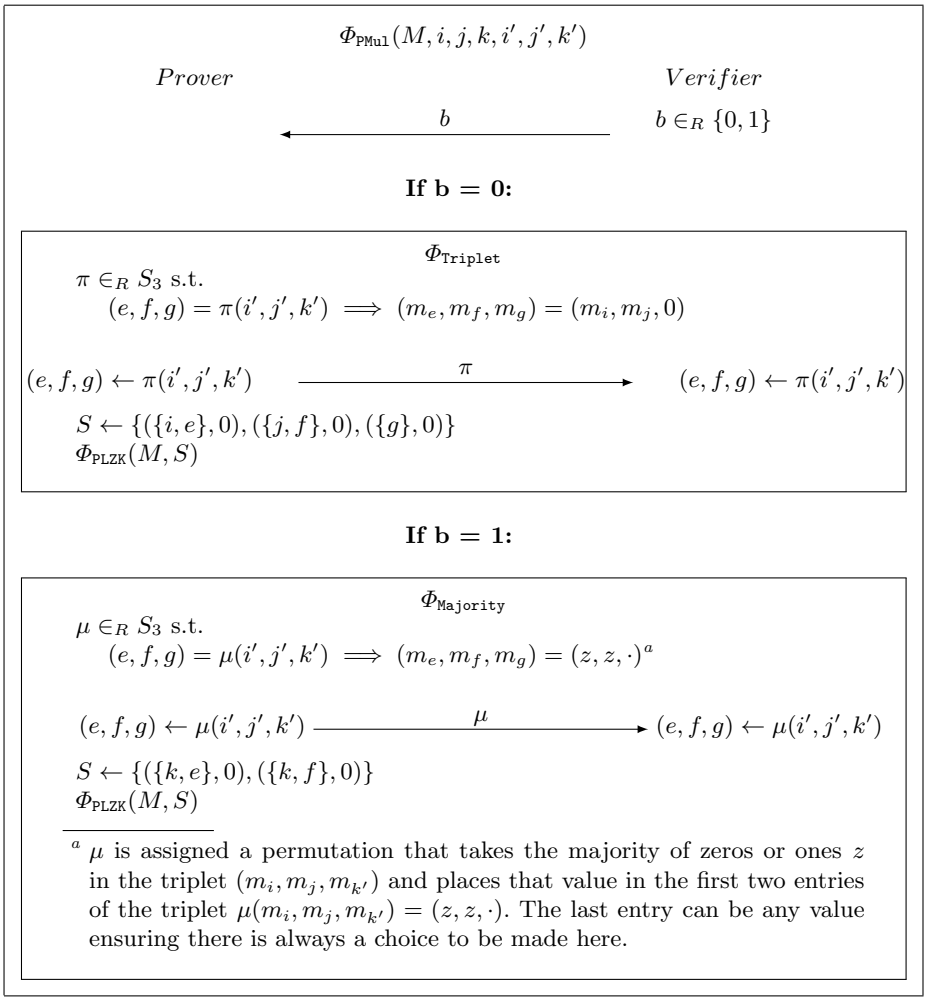


Fig. 10. Multiplication protocol

XOR gates. Essentially the Verifier commits to her challenge, then the Prover communications a set of permutations, one for each And-gate, and also prepares a set of linear relation S to be proven in parallel. In the end all the relation in S are proven using $\Phi_{\texttt{PLZK}}$.

First, the Verifier will commit to his challenge. Then both Prover and Verifier initialize a set S to be empty. The Prover will generate a string containing the input bits satisfying the circuit followed by the output bit of each gate in the circuit. Following these circuit values is some additional information: for each And-gate, the Prover generates a helper triple which are a permutation of the inputs and the 0 value. The Prover will store the permutation and also find the permutation which permutes the helper triple such that the two first elements

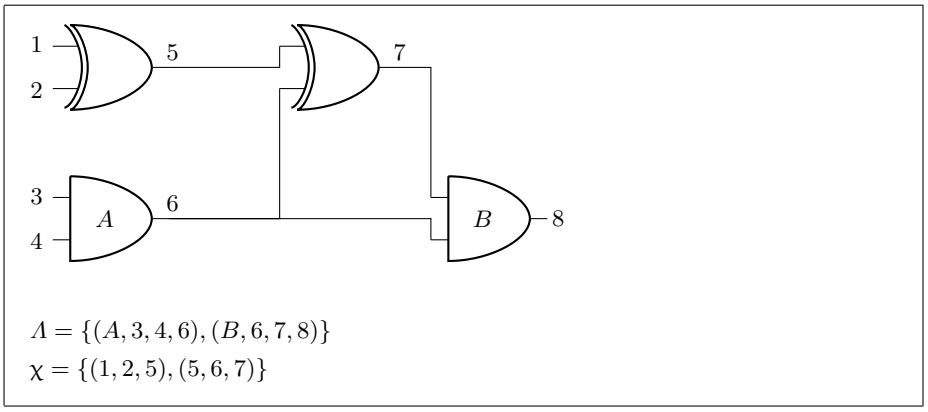


Fig. 11. Circuit representation example

of the permuted triple equal the output of the And gate. The helper triple is appended to the string.

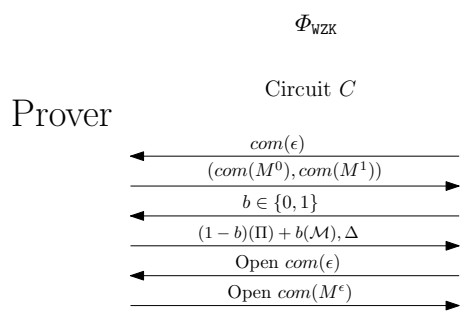
The Prover xor-commits to this string. Then, the Verifier will decide which test to perform for the And-gates. That is, the Verifier flips a coin deciding whether to check if the associated helper triple (to each And gate) is a valid permutation of its input with the value 0 or if at least two of those values are equal to the value of the output (of the And gate). The Prover
will reveal the appropriate permutations and they will add the appropriate linear tests to the set S verifying the selected property (permutation or majority equals output bit) for each And-gate. We will then insert the linear tests for Xor relations into the set S. The tests added to the S are all proved in parallel at once using $\Phi_{\mathtt{PLZK}}$. The challenge used will be the value which the Verifier committed to at the beginning of the protocol.

In full detail our protocol for circuit satisfiability goes as in Figure 12.

5.3 Generating the set S

This Section describes how the set S of linear relations in the last step of protocol $\Phi_{\mathtt{WZK}}$ is carried out given the previous steps has taken place, see Figure 12. The set S depends only on the structure of public circuit C and is generated by both parties. For each gate in C indexed by $h \in [l]$ we let i_h, j_h, k_h be the indices into the string M (and m as it is a prefix of M) corresponding to the two input bits and output bit respectively. Also, if h is an And-gate there exists an r (think the rth And-gate) for the helper triple such that $(l + 3r, l + 3r + 1, l + 3r + 2)$ denote the indices in M for the helper triple corresponding to h. Also we abuse our notation a bit and take $(e_r, f_r, g_r) = \mu_r(M_{l+3r}, M_{l+3r+1}, M_{l+3r+2})$ to be the concrete bit values (not indices) for the majority permutation. We proceed generating S as follows based on the type of gate h is[4]:

[4] The ordering of gates are are part of the public circuit description C cf. Section 5.1 and thus are the same for an honest Verifier and Prover.



Initialize The Prover and Verifier takes as input a circuit C of size $|C| = \alpha + \beta$ with α And-gates and β linear gates. The circuit will have exactly one output bit. The Prover takes additionally a *witness* x which is a satisfying assignment to the circuit input gates.

Verifier The Verifier samples challenges $\epsilon, b \in_R \{0,1\}$ uniformly at random, for the equality proofs that follows. The Verifier commits to ϵ sending $\mathtt{com}(\epsilon)$ to the Prover.

Prover [evaluation] The circuit C has an ordering of its wires such that for each index $i \in I$ m_i uniquely defines a particular wire. Thus using his witness x the Prover generates the string $m = m_0 ... m_{l-1}$ and sets m a prefix of M. That is $M_i = m_i$ for $i \in [l]$.

Prover [helper triple] For And-gates $r = 0, ..., a-1$ we have $(m_{i_r}, m_{j_r}, m_{k_r})$ s.t. $m_{i_r} \wedge m_{j_r} = m_{k_r}$. The Prover selects a random permutation $\pi_r \in S_3$ and creates a helper triple $(M_{l+3r}, M_{l+3r+1}, M_{l+3r+2}) = \pi_r(m_{i_r}, m_{j_r}, 0)$. Let $\mathit{\Pi}$ denote this set of permutations $\mathit{\Pi} = \{\pi_r\}_{r=0,...,a-1}$.

Prover [majority permutation] Also, the Prover find a *Majority* permutation μ_r such that if $(e_r, f_r, g_r) = \mu_r(M_{l+3r}, M_{l+3r+1}, M_{l+3r+2})$ then $m_{k_r} = e_r$ and also $m_{k_r} = f_r$ [a]. Let $\mathcal{M}$ denote this set of majority permutations $\mathcal{M} = \{\mu_r\}_{r=0,...,a-1}$.

Prover When Prover has formed $M = m(\pi_0(m_{i_0}, m_{j_0}, m_{k_0}), ..., \pi_{a-1}(m_{i_{a-1}}, m_{j_{a-1}}, m_{k_{a-1}}))$ of length L he selects a string $r \in \{0,1\}^L$ uniformly at random and create $M^0 = r$, $M^1 = M \oplus r$. Then he invokes the Ideal functionality $\mathrm{F}_{\mathtt{COM}}$ twice, sending an Xor commitment ($\mathtt{com}(M^0)$,$\mathtt{com}(M^1)$) to the Verifier.

Verifier Sends b in clear to the Prover.

Prover If $b = 0$ he sends $\mathit{\Pi}$ to the Verifier, if $b = 1$ he sends $\mathcal{M}$.

Prover — Verifier Both run the protocol in Figure 7 with $M, \mathtt{com}(M^0), \mathtt{com}(M^1)$ and the set $S = \{(I_r, b_r)\}_{r=0,...,l}$ as inputs with a twist: the Verifier will also open his challenge to ϵ when sending it in step 2 and the Prover checks the commitment and aborts if the open operations fails. See Section 5.3 on how S is generated.

[a] For the honest Prover such a permutation exists cf. the discussion in Section 5

Fig. 12. Zero-knowledge with Soundness one-half.

Xor gate The gate at h is an Xor gate thus we added add $(\{i_h, j_h, m_{k_h}\}, 0)$ to S proving that the three bit positions Xor to zero.

And gate The gate at h is an And gate we add either $\{(\{i_h, l+3r\}, 0), (\{j_h, l+3r+1\}, 0), (\{k_h\}, 0)\}$ or $(\{k_h, a_r\}, 0), (\{k_h, b_r\}, 0)$ depending on whether $b = 0$ or $b = 1$ in the Verifier challenge.

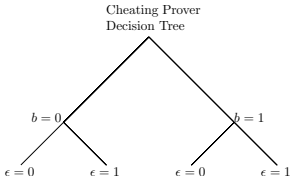
6 Weak Zero-knowledge

In this section we show our main protocol, in Figure 12, is Honest Verifier Zero-knowledge

Theorem 9. *$\Phi_{\mathtt{WZK}}$ is Honest Verifier Zero-knowledge.*

Proof. We show completeness, soundness and zero-knowledge for the protocol in Figure 12.

Completeness We will invoke $\Phi_{\mathtt{PLZK}}$ in Figure 7 in the end thus if we prove the set $S = \{(I_r, b_r)\}_{r=1,\ldots,l}$ is conflict free (e.g. for no i, j we prove $m_i = m_j \wedge m_i \neq m_j$) and stipulates the structure of C completeness follows from Theorem 8. We proceed in the structure of C. For each gate indexed by h we have corresponding $m_{i_h}, m_{j_h}, m_{k_h}$. If h points at the rth And gate it holds that $m_{i_h} \wedge m_{j_h} = m_{k_h}$ and regardless the choice of $b \in \{0, 1\}$ the Verifier accepts. To see this consider $b = 0$: We add $\{(\{i_h, l+3r\}, 0), (\{j_h, l+3r+1\}, 0), (\{k_h\}, 0)\}$ to S. That is, we check that $(m_{i_h}, m_{j_h}, 0) = \pi_r(M_{l+3r}, M_{l+3r+1}, 0)$ which is correct by construction of M. If on the other hand $b = 1$ we add to $S = (\{k_h, a_r\}, 0), (\{k_h, b_r\}, 0)$ checking that $(m_i, m_j, 0) = (z, z, \cdot)$ for $z = Maj(\pi(m_{i_h}, m_{j_h}, 0) = m_{k_h}$ which succeeds because the Prover is honest and actually does input an And gate. For Xor gates we add to the set S, $(\{i_h, j_h, m_{k_h}\}, 0)$ checking that the output bit and the two input wires of an Xor gate XOR to zero. Conflict freeness follows from C being a valid circuit and the construction of S follows the structure of C.



Soundness For a cheating Prover we show that he convinces the Verifier with probability at most $\frac{3}{4}$. For Xor gates soundness $\frac{1}{2}$ trivially follows from Theorem 8. Consider the rth And gates in C indexed by h we have $m_{i_h} \wedge m_{j_h} \neq m_{k_h}$ and the Prover will try to convince the Verifier otherwise. Since $m_{i_h} \wedge m_{j_h} \neq m_{k_h}$ both conditions $(m^i, m^j, 0) = (z, z, \cdot)$ and $(m^{i_h}, m^{j_h}, 0) = \pi_r(M^{l+3r}, M^{l+3r+1}, 0)$ will not hold at the same time. However the Prover can prepare the auxiliary triple $(M^{l+3r}, M^{l+3r+1}, M^{l+3r+2})$ such that he can win in one if the cases. Thus for either $b = 0$ or $b = 1$ the Prover is sure

to convince the Verifier and in the other case he needs to cheat the equality test which from Theorem 8 he can do with probability one-half. As b is chosen independent and uniformly at random the probability that a cheating Prover convinces a Honest Verifier is three quarters.

Zero-knowledge See the simulator in Figure 13. By the hiding property of the commitment scheme the Verifier cannot distinguish the commitments sent by the Prover from the simulated ones. The Δ will have the same distribution as in the real execution by the random choice of m. The distribution on the string M^ϵ is going to be uniformly distributed as in the real protocol but may have inconsistent bits if the Prover fails both guess on b and ϵ thus we rewind. Also if there is any Xor gates in C and the ϵ' guess failed we rewind. Otherwise the Honest Verifier will accept and the conversation is distributed as in the real case.

□

Simulator $M_{V^*}^{WZK}$

Wait for commitment The Simulator accepts $\texttt{com}(\epsilon)$.

Message The simulator picks $\epsilon', b' \in_R \{0,1\}$ and $m \in \{0,1\}^l$. It then generates a string $M = m$ and for each And gate index by h it randomly picks a permutation π_r adds a helper triple it can answer for b': if $b' = 0$ it prepares for the permutation test $(M_{l+3r}, M_{l+3r+1}, M_{l+3r+2}) = \pi_r(m_{i_h}, m_{j_h}, 0)$ otherwise it prepares for the majority test $(M_{l+3r}, M_{l+3r+1}, M_{l+3r+2}) = \mu_r(\pi_r(m_{i_h}, m_{j_h}, 0))$. It outputs $(\texttt{com}(M^0), \texttt{com}(M^1))$.

And challenge The simulator accepts b.

Permutations and Δ The simulator builds the set S following the structure of C using m as the string. However as there is no proper witness in m, some gate(s) will be inconsistent in the sense that for gate h, $m_{i_h} \square m_{j_h} \neq m_{k_h}$. For inconsistent gates the simulator build $\delta_i = \oplus_{i \in I_h} m_i^{\epsilon'}) \oplus (b_h \wedge \epsilon')$. Thus the Verifier accepts for ϵ'. Notice the number of inconsistent gates increases if the guess b' on b was wrong with all the tests for the And gates. Because now he has to provide π_r or μ_r which are inconsistent with the helper triples. However if we get the $b = b'$ these gates will be successful regardless of ϵ by construction.

Xor challenge The simulator accepts ϵ.

rewind? If both $b \neq b'$ and $\epsilon \neq \epsilon'$ then simulator rewinds V^* to the beginning. Also if the guess on ϵ was wrong and there is any Xor gates in C the simulator rewinds.

Open The Simulator opens M^ϵ.

Fig. 13. Honest Verifier Zero-knowledge simulator for Φ_{WZK}

7 Zero-knowledge for circuit satisfiability

To elevate our Weak Zero-knowledge protocol to have less soundness error we run multiple instances of the Weak protocol in parallel. We show that for security parameter κ we can construct a Universally Composable Secure protocol in the framework by Canetti [3]. We follow the style and flavour of UC advocated in [5]. We give the ideal functionality we which to realize in Figure 14.

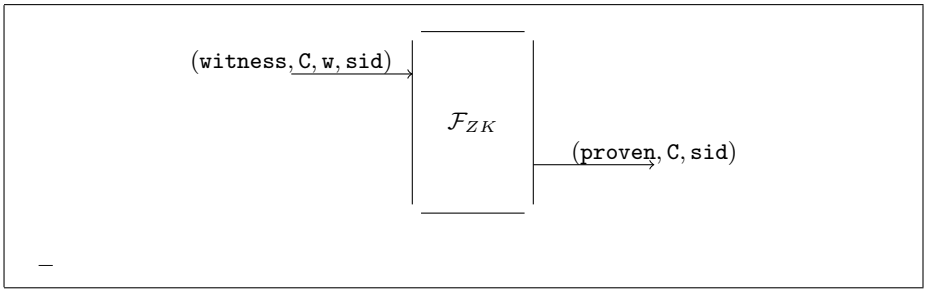


Fig. 14. Zero Knowledge Functionality

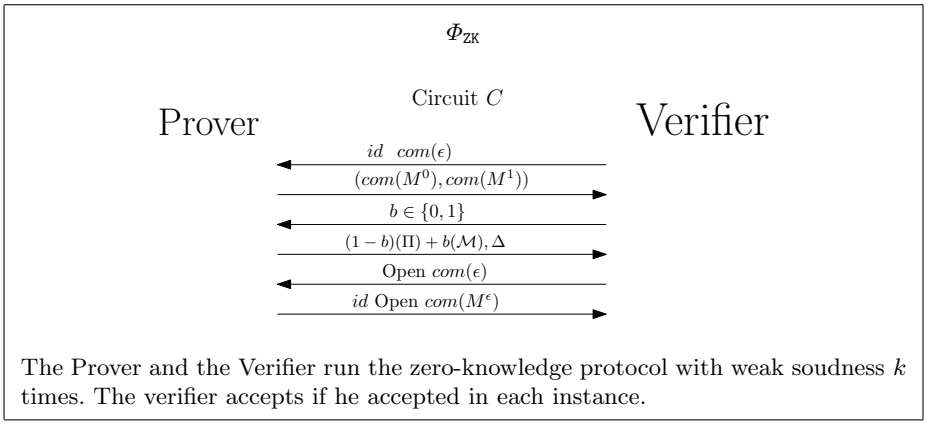


Fig. 15. Universally Composable Secure Protocol for Zero Knowledge

Theorem 10. *$\Phi_{\mathtt{ZK}}$ securely realizes $\mathrm{F}_{\mathtt{ZK}}$ in the $\mathrm{F}_{\mathtt{COM}}$ hybrid model for any static, active adversary.*

Proof. We construct a simulator $\mathrm{S}_{\mathtt{ZK}}$ such that every poly-time environment $\mathcal{Z}$ cannot distinguish between the real protocol system $\mathrm{F}_{\mathtt{COM}}$ composed with $\Phi_{\mathtt{ZK}}$ or $\mathrm{F}_{\mathtt{ZK}}$ composed with $\mathrm{S}_{\mathtt{ZK}}$. The Simulator will play the role of the honest party. We give below two (sub)simulators one for a corrupt Prover and one for a corrupt

Verifier. S_{ZK} will determine which to run when the environment $\mathcal{Z}$ stipulates which player is corrupt.

7.1 Corrupt Prover

The simulator proceeds in two steps.

First, the simulator runs each instance of the weak zero-knowledge protocol in parallel: He sends the committed message for the commitment of the challenge to the environment. The simulator awaits the commit messages for each of the xor-commitments. The simulator then selects a random bit which selects the permutation or majority test, records it and sends it to the environment. The simulator records the value sent to him for the given test. The simulator then selects a random $\epsilon \in_R \{0,1\}$ and sends a reveal command for the commitment functionality for the given ϵ. The simulator awaits that the environment sends the reveal command intended for the M^ϵ. The simulator then records if a given instance would result in a verifier accepting or rejecting.

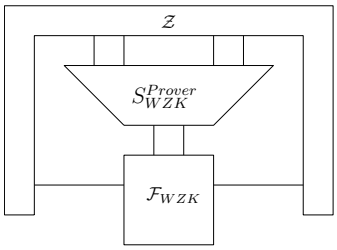
Second, if for any instance, a verifier would reject. The simulator aborts. Otherwise, he scans each instance looking for one where the substring associated to the circuit contains a valid assignment to the circuit which has output one. If no witness is found the simulator aborts. The simulator selects one of those inputs and feeds it to F_{ZK}. This completes the description of the simulator.

We will proceed to show that $S_{ZK} \circ F_{ZK} \overset{\text{stat}}{\equiv} \Phi_{ZK} \circ F_{COM}$. We can see that the messages sent by the verifier to the prover in the real world are perfectly indistinguishable from those sent by the simulator to the environment. The only thing the environment can use to distinguish between these two cases is the output. Now there are three cases, when an abort occurs due to rejection in an instance, when a valid witness is provided for one of the instances and finally when no witness is provided. When an abort occurs due to a failed test, the probability that this view would have been generated is the same in both worlds. The same applies when a witness is provided in one of the xor-commitments. The only case that the views are distinguishable is when the environment passes each instance without providing a valid witness. Fortunately, by soundness of the weak zero-knowledge protocol, this only occurs with negligible probability. Thus, we have that the views are statistically indistinguishable.

7.2 Corrupt Verifier

On receiving the message `proven` from the ideal functionality the simulator runs prepare to simulator many instances of the Prover. He awaits the commit command for ϵ. The simulator then sends the commit commands for the xor-commitments. He awaits the choice of test from the environment. With knowledge of both ϵ and the choice of test, the simulator can construct Δ and permutations which an honest verifier would accept. When finally the environment sends the open command for ϵ, the simulator sends the reveal message for the given commitment to the receiver. By Theorem 3, the string revealed looks independent in the real world. Thus, the distribution of the revealed string is the

same in both the real and ideal world. As a result, the real and ideal worlds are indistinguishable. An overview of the Simulator is given in Figure 17. □



The Simulator S_{ZK}^{Prover} creates the view a corrupt Prover sees for the environment $\mathcal{Z}$ which is indistinguishable from the real protocol execution, while providing the input of the Prover to the ideal functionality.

Initialize Upon receiving a circuit C from the environment the Simulator samples $\epsilon \in_R \{0,1\}$

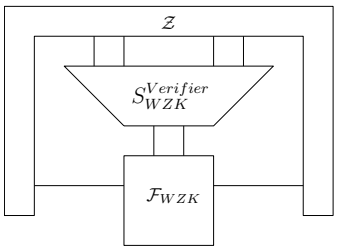
Commit The simulator honestly commits to ϵ creating $\mathtt{com}(\epsilon)$.

Prover commits The environment provides commitments $\mathtt{com}(M^0)$ and $\mathtt{com}(M^1)$ to the simulator. By extractability of the ideal commitment functionality $\mathrm{F}_{\mathtt{COM}}$ the message $M = M^0 \oplus M^1$ is computed by S_{ZK}^{Prover}.

Follow protocol After this step the simulator complete the protocol with the environment.

Invoke Ideal $\mathcal{F}_{ZK}$ with witness w which is a prefix of M. The ideal functionality outputs to the environment the message for the honest Verifier.

Fig. 16. Simulator in case of a corrupted Prover



The Simulator $S_{ZK}^{Verifier}$ creates the view a corrupt Verifier sees for the environment $\mathcal{Z}$ which is indistinguishable from the real protocol execution given the $(\texttt{proven}, C, pid)$ output from the ideal functionality.

Initialize Upon receiving $(\texttt{proven}, C, sid)$ from the ideal functionality the Simulator computes $L = l + 3\alpha$, where l is the number of wires in C and α is the number of And-gates in C.

Accept Challenge Commitment Upon receiving $\texttt{com}(\epsilon)$ from the environment, the simulator stores ϵ. The simulator picks $M^{\epsilon} \in_R \{0,1\}^L$, and to generate a $\Delta = \delta_1, ..., \delta_l$ string that it can answer it does the following: First it generates the set $S = \{(I_v, b_v)\}_{v-1,...,l}$ as in Section 5.3. Then it computes $\delta_v = \bigoplus_{i \in I_v} M_i^{\epsilon} \oplus (b_v \wedge \epsilon)$. The simulator set $M^{1-\epsilon} = 0^L$.

Commit The Simulator sends $(\texttt{com}(M^0), \texttt{com}(M^1))$ to the environment.

Completes the protocol The simulator completes the protocol with the environment.

Fig. 17. Simulator in case of the corrupted Verifier

8 Empirical studies

The protocol has been implemented in C for fine grained control over the Big-Oh constants. The implementation is available at
http://tinyurl.com/om6vvh6

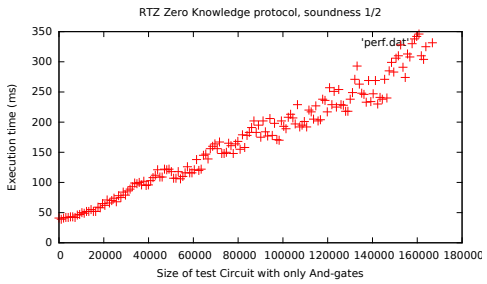
Our test setup is two machines acting as Prover and Verifier connected on a GigaBit ethernet network of our department. Our machines has 8 GigaBytes of memory and Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz with 8 cores.

Our primary benchmark application is that of AES. We have an optimized AES-circuit[5] with 6800 And-gates and 26816 linear gates taking 50 ms with soundness one-half. Communication complexity is 4 bits per And-gate and 1 bit per linear gate for soundness one quarter. Below we see execution time increase with the number of And-gates.

9 Conclusions

We have presented an information theoretic construction extending commitments to zero knowledge proofs. We show that when our construction builds

[5] Thanks to Nigel Smart and his team at Bristol, https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/.



on top of a UC-secure commitment scheme it is UC-secure. The flavor of supported proofs are circuit satisfiability. The statement to be proven is a circuit for which the Prover knows a satisfying assignment for the input gates. On the theoretical side our scheme exhibits small constants in addition to the underlying commitment scheme.

We have an implementation of our scheme using UC commitments in the Random Oracle Model, where the oracle is implemented by SHA-512. We can prove knowledge of an AES-key encrypting a particular plaintext to a particular ciphertext in less than 500 milliseconds with soundness error $\frac{1}{2^{40}}$.

Acknowledgement

We would like to thank Claudio Orlandi and Irene Giacomelli for reading early drafts giving valued comments and for useful discussions.

References

1. Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *Advances in Cryptology-CRYPTO 88*, pages 37–56, 1988.
2. Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, pages 156–189, 1988.
3. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/.
4. Ignacio Cascudo, Ivan Damgård, Bernardo Machado David, Irene Giacomelli, Jesper Buus Nielsen, and Roberto Trifiletti. Additively homomorphic UC commitments with optimal amortized overhead. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, pages 495–515, 2015.
5. Roanld Cramer, Ivan Damgård, and Jesper Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 1st edition, July 2015.
6. Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In *Advances in Cryptology—CRYPT0 95*, pages 110–123. Springer, 1995.
7. Ivan Damgård, Bernardo Machado David, Irene Giacomelli, and Jesper Buus Nielsen. Compact VSS and efficient homomorphic UC commitments. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, pages 213–232, 2014.
8. Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 501–520. Springer Berlin Heidelberg, 2006.
9. Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. Cryptology ePrint Archive, Report 2014/598, 2014. http://eprint.iacr.org/.
10. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.
11. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.
12. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing*, pages 186–208, 1989.
13. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30. ACM, 2007.
14. Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 955–966, 2013.
15. Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

16. Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732. ACM, 1992.
17. MichaelO. Rabin, Yishay Mansour, S. Muthukrishnan, and Moti Yung. Strictly-black-box zero-knowledge and efficient validation of financial transactions. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, volume 7391 of *Lecture Notes in Computer Science*, pages 738–749. Springer Berlin Heidelberg, 2012.
18. Alain Tapp. Cryptography based on oblivious transfer. In *Proceedings of Pragocrypt 96*. 1996.

A Reproducing our empirical studies

We have implemented our protocol described in Figure 12 for Weak Zero-knowledge. The implementation can be found at
http://tinyurl.com/om6vvh6

A.1 Software structure

The software project is written from scratch using only few dependencies on the system like some libstdc functionality. We do this in order to have fine grained control of the performance of our program. The structure is as follows:

`platform` Inside the platform directory we have all the OS/HW dependent code
`common` Inside common we have library code needed to implement the protocol, including network management in `CArena` and data-structures in project `ds`.
`empiricalZK` holds two projects: `RTZ14` which is the code for protocol described in this paper. `IKOS` will later be populated with an efficient implementation of the MPC in the HEAD idea which is in its infant stage right now. We wish to publish a comparison between IKOS and RTX14 (this protocol) in a follow up paper.

All projects are GNU Auto-Make/Conf projects producing a static library and some also an executable. Each project defines a configuration item with version control for maintenance.

A.2 Dependencies

The code is written with in *C* for speed and portability. It includes work by Nayuki Minase published at http://www.nayuki.io/page/fast-sha2-hashes-in-x86-assembly.

The build system on FreeBSD 10, OSX and GNU Linux requires:

GNU Bash 4.3.11(2)
Automake 1.14
Autoconf 2.69

Or on Windows 8/10 a working Community version of Visual Studio Express 2013 or later is required.

A.3 Getting the Source

Install git on your system and do

```
git clone http://tinyurl.com/om6vvh6
(you may need to replace this by the actual url).
```

A.4 Building from Source code, FreeBSD, Linux and OSX

On these systems building the source is done by changing directory to where you have checked out the source and locating the `build.sh` script.

```
user@host $ ./build.sh release
```

Will build the prover executable in empiricalZK/rtz14/linux/src/prover.

A.5 Building from Source code, Windows 8/10

On Windows we have a test solution that as a bi-product of running the test programs also produces the prover.exe in empiricalZK/rtz14/win64/rtz14/Release/prover.exe.

You can run this executable from a Command Prompt invoking it with no argument to see your options and for running it providing arguments to do a Zero-knowledge proof.

A.6 Reproducing our results

Our benchmark application is proving knowledge of a particular AES key given a public plaintext and ciphertext. The structure of the circuit we prove to satisfy is depicted in Figure 18. The circuit includes public constant assignments for the plaintext and the Prover convinces the Verifier that he has knowledge of an Aes-Key encrypting this particular plaintext to a ciphertext built into the circuit. That is, our binary AES is extended with the top-triangle on Figure 18 which is a small comparison circuit with public constants stipulating the expected ciphertext and comparing with the output of the AES circuit (the larger triangle below it). In the end all the Verifier learns is that the prover has a witness w making the (public) circuit true, thus encrypting the given plaintext to the expected ciphertext.

Our lab computers has the following specifications:

```
CPU: i7-3770K CPU @ 3.50GHz with 8 cores
Mem: 8Gb of Ram
Net: Gigabit LAN
OS:  3.13.0-59-generic #98-Ubuntu SMP Fri Jul 24
     21:05:26 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
```

for both Verifier and Prover. On the machine intended for the Prover do

```
./rtz14 -circuit ../test/AES -witness 00000000000000000000000000000000
        -port 2020
```

This will start a prover process listening for a Verifier to connect. The circuit in ../test/AES is the following and we prove in this case that we have knowledge of a key (this witness above which is the zero key) encrypting the all zero plaintext (that is 16 zero bytes) to the AES ciphertext under the zero key, namely:

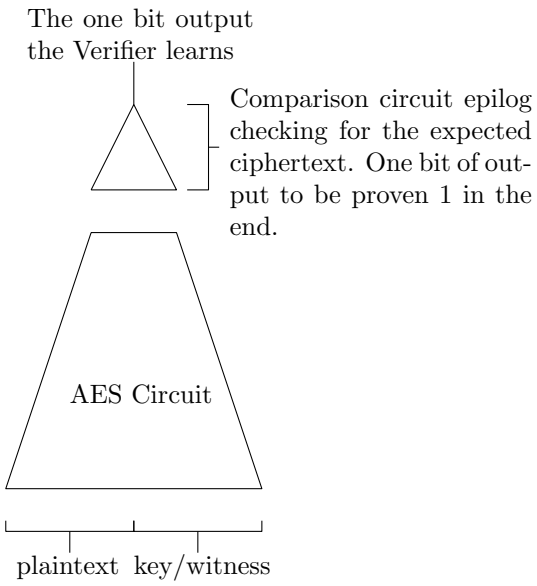


Fig. 18. A modified binary AES circuit for proving knowledge of a key encrypting a particular plaintext to a particular ciphertext.

```
0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 1
1 1 0 1 0 0 1 0 0 0 1 0 1 0 1 1
1 1 1 1 0 1 1 1 0 1 0 1 0 0 0 1
0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 0
0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0
0 1 0 1 1 1 1 1 1 0 0 1 1 0 1 0
0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0
1 1 0 1 0 1 0 0 0 1 1 1 0 1 0 0
```

To start the Verifier you need the ip address of the Prover, assuming it is `xxx.yyy.zzz.www` execute the following on the Verifier Machine:

```
./rtz14 -circuit ../test/AES -port 2020 -ip xxx.yyy.zzz.www
```

The process uses the -witness argument to distinguish whether to run as Prover or Verifier. This proves with error probability 3/4 that the prover knows such a witness. Because of completeness this will always succeed if the Prover inputs the correct witness, otherwise the Verfier only accepts with a 75% probability. In a real word application the protocol will be repeated 3κ times (for $\kappa = 256$) to ensure $2^{-\kappa}$ probability that a cheating Prover wins. Our experiment above runs in $3ms$ on our test machines thus for security parameter κ we expect a real world running time of $9\kappa ms \approx 2.5$ seconds.

B Concrete example: ZK-proof that $(x_1 \wedge x_2) \oplus (x_3 \oplus x_4)$ is satisfiable

We will give an example here where a Prover proves in Zero-knowledge that he knows a satisfying assignment for the circuit $(x_1 \wedge x_2) \oplus (x_3 \oplus x_4)$. In fact the steps illustrated by this example is those of Φ_{PMul} in Figure 10. We will abuse notation a bit for a cleaner presentation. Openings of all protocols invoked in the follow should be pick up and done in the very end. That is, when we invoke e.g. the protocol for equality the *xom* commitment is opened. However this should in a real execution be postponed to the very end for the proof of the entire circuit.

B.1 Step 1:

Generate a string a string w of length $n + 3a$. All bit positions in this string are set to $\perp$ initially. Recall n is the number of input wires and gates all together in the circuit. a is the number of and gates are present in the circuit. We will need the Prover to commit to some helper triples for each and gate. As we have 4 input wires, 3 gates one of which are an and gate we have our string w of length 10, as $n = 7$ and $a = 1$. $w = (\perp, \perp, \perp, \perp\perp, \perp, \perp, \perp, \perp, \perp)$.

B.2 Step 2: Generate satisfying assignment with output one

In the first step, the Prover generates a satisfying assignment for the circuits. In essence he selects a valid assignment for the circuit which outputs 1. Input these values in the string.

- $x_5 = x_1 \wedge x_2$
- $x_6 = x_3 \oplus x_4$
- $x_7 = x_5 \oplus x_6$

- $x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 0$
- $x_5 = 1,\ x_6 = 0$
- $x_7 = 1$

- foreach $i \in \{1, \ldots, 7\}$, $w_i \leftarrow x_i$

$w = (1, 1, 0, 0, 1, 0, 1, \perp, \perp, \perp)$

As there is one And gate in the circuit we have left room for at one helper triple, that what the three $\perp$ are placeholders for.

B.3 Step 3: Append permutation of inputs for And gates and include permutations in set

In the second step, for each And triple, the Prover generates a random permutation π. The only And triplet is of the form $(p, i, j, k) = (1, 1, 2, 5)$ where p is the index of the And triple, i, j are the indices of the input operands to the And gate and k is where the result is. That is the Prover prepares to convince the Verifier of the And relationship $m_1 \wedge m_2 = m_5$ by appending a random permutation as follows:

- suppose π was randomly selected to be $\pi(1,2,3) = (2,3,1)$
- $(d_1, d_2, d_3) \leftarrow (2,3,1) = \pi(1,2,3)$
- $(w_{z+d_1}, w_{z+d_2}, w_{z+d_3}) = w_{z+2}, w_{z+3}, w_{z+1} \leftarrow (w_1, w_2, 0)$
- μ will be chosen such that $\mu(1,2,3) \in \{(2,3,1),(3,2,1)\}$
- Prover sets $\Pi \leftarrow \Pi \cdot (c, \pi)$
- Prover sets $P \leftarrow P \cdot (c, \mu)$

$w = (1,1,0,0,1,0,1,1,1,0)$

B.4 Step 4: use commitment with linear proofs to commit to circuit and helper triples

In the third step, the Prover xor commits to this constructed message w with the circuit and the helper triples. That is he applies $\Phi_{commit}(w)$ such that both Prover and Verifier obtain:

- $W = xom(w) := (com(w^0), com(w^1))$

B.5 Step 5: Permutation test selection

In this step, the Verifier selects which permutation test will be used for the multiplicative proof. In particular, will the Verifier check that $(m_{n+1}, m_{n+2}m, m_{n+3})$ is a permutation of $(m_1, m_2, 0)$ or that a received permutation $p \in S_3$, such that for $d_1, d_2, d_3 := \pi(1,2,3)$ that such that $m_5 = m_{n+d_1} \wedge m_5 = m_{n+d_2}$. In this case, it would be $m_5 = m_{10}$, $m_5 = w_9$ (since $m(1,2,3) = (3,2,1)$)

- The Verifier selects which test to do

B.6 Step 6: Prover circuit commitment

This step combines the first part of the equality test for the and triples and the equality tests. The Prover will reveal the δ for the different proofs of equality. This forces -due to the proof of equality- a dishonest Prover to decide which challenge he can't respond to. If we think of this protocol as a sigma protocol, this is the point where the Prover has finished committing to his values.

1. If triplet challenge was selected
2. The Prover sends the permutation π of $(w_1, w_2, 0)$
3. The Prover and Verifier now executes $\Phi_{Eq}(W, 1, 9)$ $\Phi_{Eq}(W, 2, 10)$ $\Phi_{Zeq}(W, 8)$. If all three tests passes the Verifier accepts.

1. If the Majority challenge was selected
2. Prover sends μ e.g. say $(2,3,1)$ to Verifier.
3. Now since the helper triple is in index 8,9,10 this permutation translates into the triple (w_9, w_{10}, w_8)
4. The Prover and Verifier now executes $\Phi_{Eq}(W, 1, 9)$ $\Phi_{Eq}(W, 2, 10)$ and the Verifier accepts if both equality tests passes.

B.7 Step 6: The linear gates

We still need the Prover to prove the two Xor gates in the circuit. That is we need to prove $w_3^0 \oplus w_4^0 \oplus w_6^0 = w_3^1 \oplus w_4^1 \oplus w_6^1$ and that $w_5^0 \oplus w_6^0 \oplus w_7^0 = w_5^1 \oplus w_6^1 \oplus w_7^1$.

This can be done using the generalized equality box $\Phi_{\texttt{LZK}}(W, \{(I_1, 0), (I_2, 0)\})$ where $I_1 = \{3, 4, 6\}$ and $I_2 = 5, 6, 7$.

Recall the circuit is public and at this point the Prover has shown the relationships between bit positions of the circuit encoded in w. Thus the Verifier is convinced that w_{10} will hold the result.

B.8 Step 7: The final step

In the final step the Prover shows that w_{10} equals one.

C Inequality protocol and proofs

$M = \texttt{xom}(m) = (\texttt{com}(m_0), \texttt{com}(m_1))$ Prover inequality between two bits of the committed string.

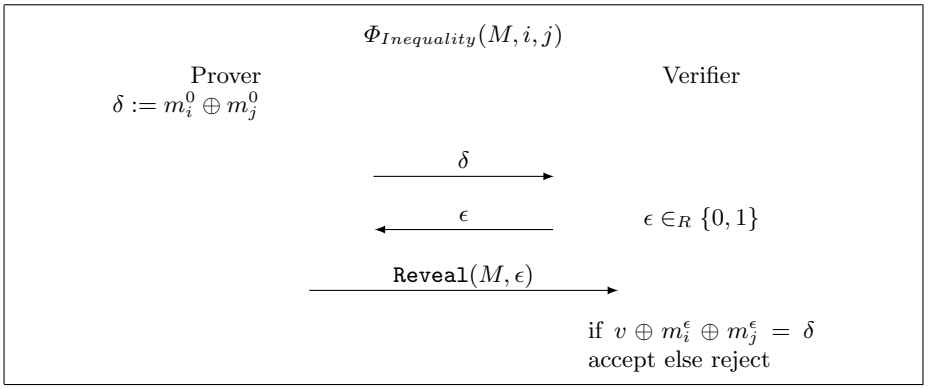


Fig. 19. Inequality

C.1 Zero-knowledge

Completeness: Since $m^i \neq m^i$, we can see that
case 1: if $m_0^i = m_0^j$ then $m_1^i \neq m_j^1$ in which case $\epsilon = 0 = m_0^i \oplus m_0^j = m_1^i \oplus m_j^1 \oplus 1$.
case 2: if $m_0^i \neq m_0^j$ then $m_1^i = m_j^1$ in which case $\epsilon = 1 = m_0^i \oplus m_0^j = m_1^i \oplus m_j^1 \oplus 1$.

Soundness: if $x = y$ then $m_0^i \oplus m_0^j = m_1^i \oplus m_j^1$ and therefore for any value of ϵ there exists a value v such that the Verifier will not accept. As such a cheater is detected with probability $1/2$.

Honest Verifier Zero-Knowledge: To prove the protocol is zero-knowledge for a Verifier we give the description of a simulator $M_{V^*}^{Neq}$ that generates a view (δ, m^b) indistinguishable from a real execution.

Simulator $M_{V^*}^{Neq}(i, j)$

1 Pick at random $m, r \in \{0,1\}^l, m^i \neq m^j$.
2 Compute $\delta = m_i^0 \oplus m_j^0$ and output $(\texttt{com}(m^0), \texttt{com}(m^1), \delta)$ to the Verifier V^*.
3 Accept $\epsilon \; in \{0, 1\}$.
4 Open $\texttt{com}(m^\epsilon)$

Fig. 20. Simulator for Linear Zero Knowledge

The Simulator follows the protocol with no knowledge of a particular witness w. $\square$