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.

1 Introduction

Zero-knowledge was introduced in 1985 by Goldwasser, Micali and Rackoff in their seminal paper [6] 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.

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

* 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

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 to NP-complete languages.

Generic constructions for Zero-knowledge are starting to emerge. The recent line work starting with [8] 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 [4] where they build specialized garbling schemes tailored for Zero-knowledge proofs. The garbling approach communicates at least one symmetric encryption per and gate in the circuit. In contrast, our protocol only requires a number of commitments proportional to the security parameter where the constant is small (roughly 5).

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. Such a scheme can be realized in the Random Oracle Model in a very efficient way. This makes our construction extremely efficient, since it only relies on evaluating hash functions.

We take a similar approached then the one used in [2,3]. They use xor operations over individually committed bits to prove statements. We employ strings which allow us to get faster protocols.

In a bit more detail, our construction is a novel way of committing to bit-strings, enabling 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 is one and this is essentially the only new information that the verifier learns.

For a circuit of size n with l input gates, a and gates and g linear gates our construction communicates $6a + g$ bits of data with soundness one-half. To form a secure protocol with security $2^{-\kappa}$, we repeat our construction κ times realizing a protocol with communication complexity $O(\kappa n)$. We emphasize 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 bits within a committed string. These relationships include equality, inequality and a proof that a set of committed bits 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 $\text{xom}(m)$ to say that a prover commits to a message m using commitments with linear proofs.

The proofs will take the form of sigma protocols. 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.

The xor commitment scheme is as follows. The commit phase has two-steps. First, the prover will choose a string r uniformly at random, he will then commit to $m^0 = r$ and he will also commit to $m^1 = m \oplus r$ using a standard commitment scheme from the literature denoted by $\text{com}(\cdot)$.

The first important property of this commitment scheme is that for any committed value m, the view of m^i is indistinguishable from a uniformly random string as long as m^{1-i} stays hidden.

If two bit positions in m are equal, say $m_i = m_j$, then there exists a δ such that $\delta = m_i^0 \oplus m_j^0 = m_i^1 \oplus m_j^1$. The view (δ, m^i) looks independent of m as long as m^{1-k} stays hidden.

In contrast, if $m_i \neq m_j$, then for any choice of δ, $\delta \neq m_i^0$ or m_j^0 or $\delta \neq m_i^1 \oplus m_j^1$. This is equivalent to saying there exists an ϵ such that $\epsilon = m_i^0 \oplus m_j^0 = 1 \oplus m_i^1 \oplus m_j^1$. The view (ϵ, m^k) looks independent of m as long as (m^{1-k}) remain hidden.

These properties will allow us to generate proofs for linear relationships among commitments.

3.1 Xor commitment scheme

Let $m \in \{0,1\}^l$ be a bit string of length l. To create an $\text{xom}(\cdot)$ commitment to m a random message $r \in_R \{0,1\}^l$ is chosen. Then we compute $m^0 = r$ and $m^1 = m \oplus r$ and define $\text{xom}(m) = (\text{com}(\text{m}^0), \text{com}(\text{m}^1))$ see Figure 1.

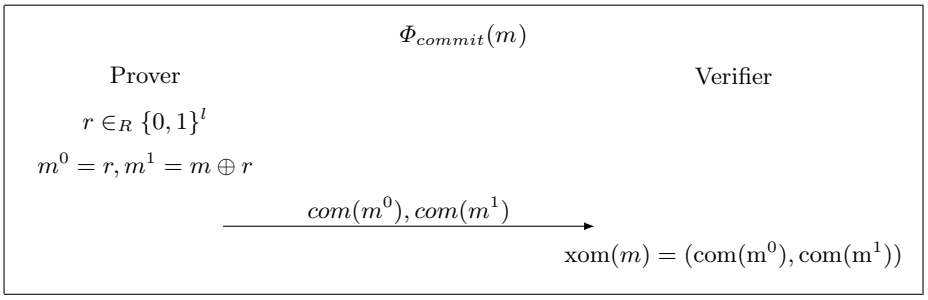

Fig. 1. Commit

3.2 Reveal

To open completely, we simply open both commitments. 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 the xor commitment. For a commitment with linear proofs M, associated to value m with standard commitments we will use $\texttt{Reveal}(M, t)$ to denote a prover opening the value of the standard commitment $\mathrm{com}(\mathrm{m}^{\mathrm{t}})$.

4 Zero Knowledge with weak soundness

In our first result we show how to do an honest verifier zero knowledge proof of equality between two bit positions of an xor-commitment to a string m. The basic idea is as follows: we will exploit the fact that if the bits m_i, m_j in positions i, j of m 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 $m_i \neq m_j$ then no such δ exists.

We can use this fact to prove equality of the two bit positions in m revealing essentially nothing else. Informally, observe that for any $b \in \{0,1\}$, the view (δ, m_i^b, m_i^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. In the first step of the proof, 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.

4.1 Protocol for equality

For a string $m \in \{0,1\}^l$ and $M = \mathrm{xom}(m) = (\mathrm{com}(\mathrm{m}_0), \mathrm{com}(\mathrm{m}_1))$, we show how to prove for a given i, j that $m_i = m_j$. The scheme allows the prover to prove equality for multiple positions $\{(i_k, j_k)\}_k$, however recall that the challenge ϵ used has to be the same otherwise the bits m_i and m_j are revealed. The protocol for equality is depicted in Figure 2.

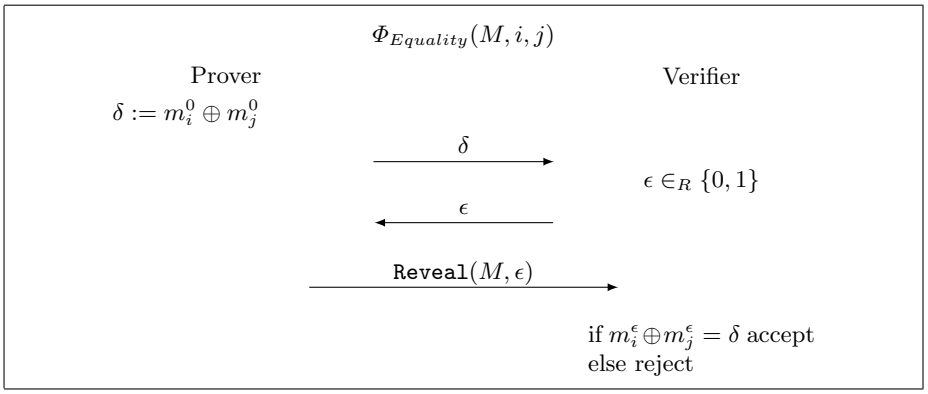

Fig. 2. Equality

4.2 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 verifier 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 given that we are only proving equality 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, if $m_i \neq m_j$ and the prover will 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. As 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 (δ, m^b) which is indistinguishable from a real execution, see Figure 3.

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

1 Select $b \in_R \{0, 1\}$
2 Compute $\delta \leftarrow m_i^b \oplus m_j^b$
3 Accept to receive the $\epsilon \in \{0, 1\}$
4 If $\epsilon = b$ the simulator continues and opens the commitment to the message m^ϵ.
5 Otherwise we rewind to step 1.

Fig. 3. Simulator for equality

Consider the simulator in Figure 3. We argue that the view generated is indistinguishable from a real execution: clearly the distribution of δ send to the verifier is uniform as in the real execution. Also, in Step 4, the simulator behaves exactly as in the real execution and the transcript exhibits exactly the same distribution. From this, we conclude that the view (δ, m^b) is indistinguishable from a real execution. □

4.3 Parallel equality proofs

For an xor-commitment $M = \mathrm{xom}(m)$ and for a set of pairs of indices $\{(i_v, j_v)\}_{v=1,\ldots,t}$ into m we can prove all positions equal with soundness one-half.

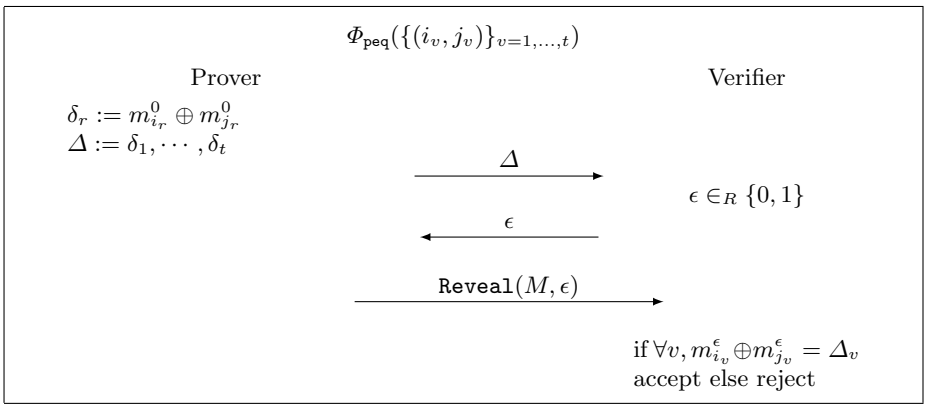

Fig. 4. Parallel Equality

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 since it follows trivially from the definition.

5 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 $b \in \{0,1\}$, $\epsilon = m^i_b \oplus m^j_b \oplus b$ instead if the $\delta = m^0_i \oplus m^0_j$ as before. We put the protocol and its proof in the appendix.

6 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. That is, protocols where a prover commits to a message and convinces a verifier that certain bit positions in the committed

message are equal bits or different bits. In fact we can combine these two in one protocol to convince our verifier that bit positions XOR to a particular value. This cover equality and inequality as special cases.

The input of the protocol will be a pair of elements, the first element will be a set of indices and the second will be the value that they are supposed to xor to. Note that equality is covered by putting two elements in the set and setting it to 0 and by putting a one instead we have inequality. In addition, by only putting a single index in the set, it is a proof that the value committed is equal to the given bit without having to open both strings. Figure 5 below depicts our protocol for proving the XOR relation between bit positions. Figure 6 is a protocol showing how to pack multiple such proofs together into one protocol still using only one xom(·).

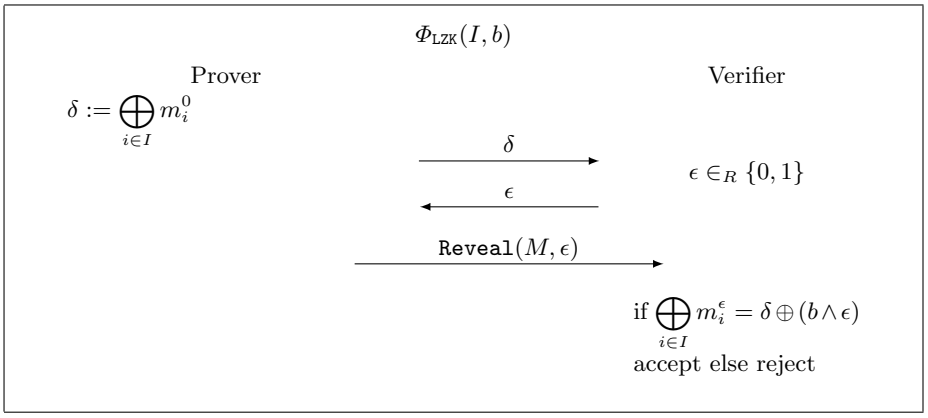

Fig. 5. Linear Zero-knowledge

Secure Proofs Our proofs of equality, inequality and linear relations above has soundness one-half. For a security parameter κ we later show how we can make a protocol that essentially repeats the proofs of equality κ times to decrease the success probability of a dishonest prover to $2^{-\kappa}$.

6.1 And-proof

In this section, we wish to be able to prove that for an xor-committed string 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 another 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

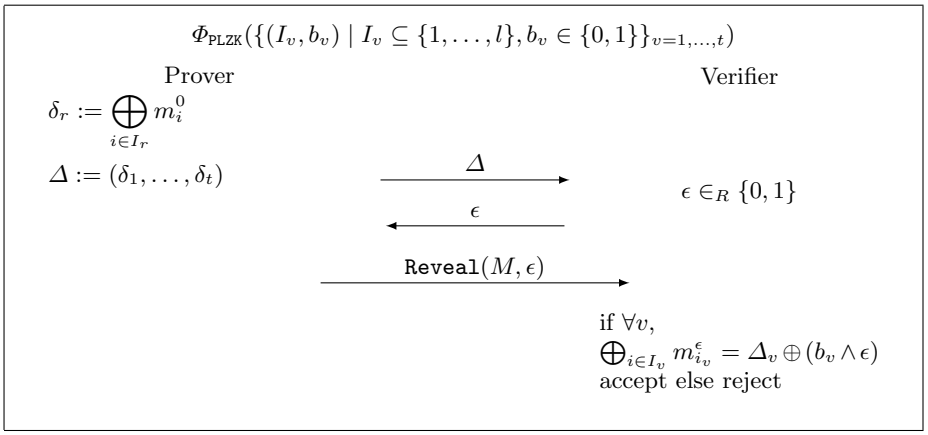

Fig. 6. Parallel Linear Zero-knowledge

depicted in Figure 7 and proceeds as follows: the prover will for each and gate, randomly ask that the receiver 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 output value. If the bits do not form a valid triple then the prover must fail one of two tests with probability one-half. As a result, a cheating prover will get caught with probability one-quarter.

The proof that this protocol is honest verifier zero knowledge follows from a later proof where we show this protocol is UC-secure if the underlying commitment scheme is UC-secure. That is, we show the verifiers view can be simulated and for UC-secure commitments the provers side can also be simulated. We refer to Section 8.

7 Zero-knowledge for circuit satisfiability

In this section we will give a zero-knowledge proof of circuit satisfiability. This is done by combining the equality, inequality, and the and test together. Notice that if we commit to a satisfying assignment for the circuit using an xor-commitment, we can already prove linear relations. This means that with the and proof we get a Zero-knowledge proof of satisfiability for any circuit.

7.1 Circuit notation

In this section, we define a circuit notation which is convenient for our zero-knowledge proofs. Indices will be used to identify 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. A negation gate is represented as a pair of indices where the first value of the pair is the index of the input wire and the second value is the index of the output wire. In a similar fashion, the xor gates will be represented as a triple of values

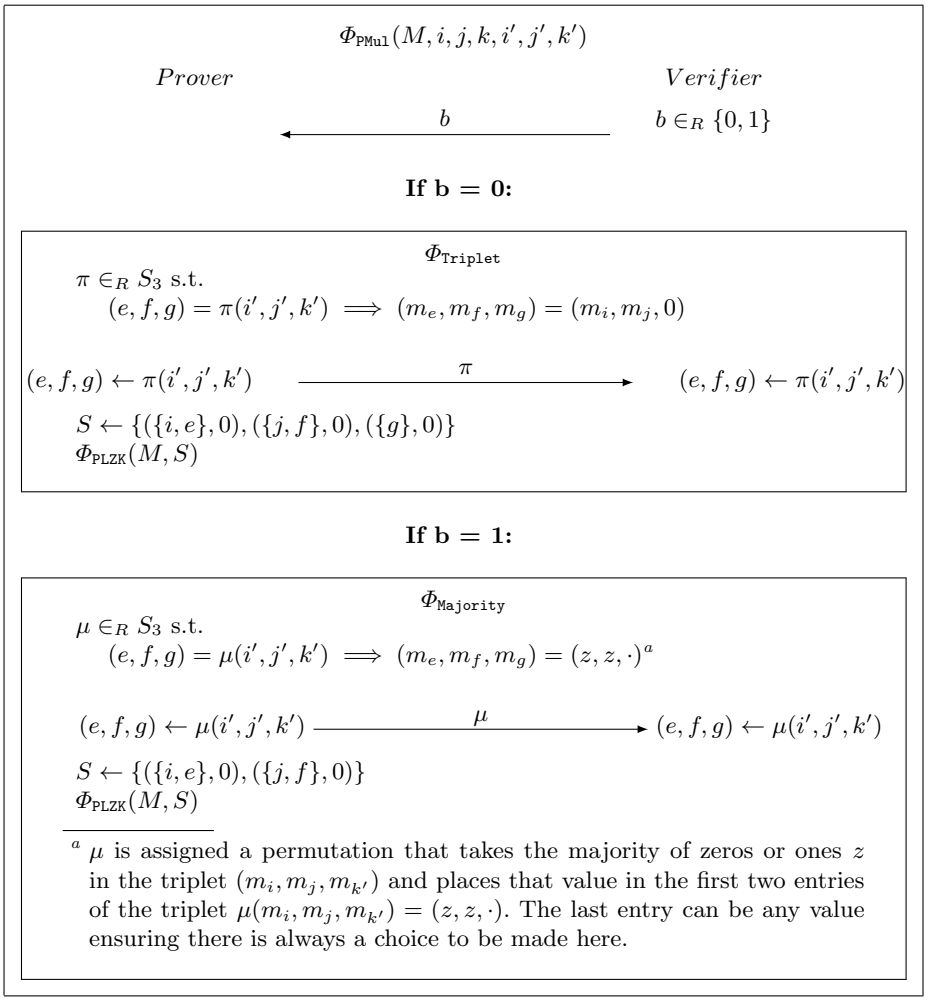

Fig. 7. Multiplication protocol

where the first two values will be the index of the input wires and the third will be the index of the output wire. Finally, for the and gates, each and gate will consist of 4 indices, the first value will be a unique identifier given to the and gate, the second and third values will be the input wires and the fourth value will be the index of the output wire. The identifier given to the and gates will allow us to reference them in our protocol and will also be used to find the associated helper triples. We will denote Λ as the set of and gates, χ as the set of xor gates and $\mathscr{N}$ as the set of negation gates. We denote n as the number of wires and a as the number of and gates. We denote Π as a set of pairs (c, π) where $1 \leq c \leq a$ and $\pi \in S_3$ which is the set of permutations of three elements. We define v as the values of a valid assignment for the circuit.

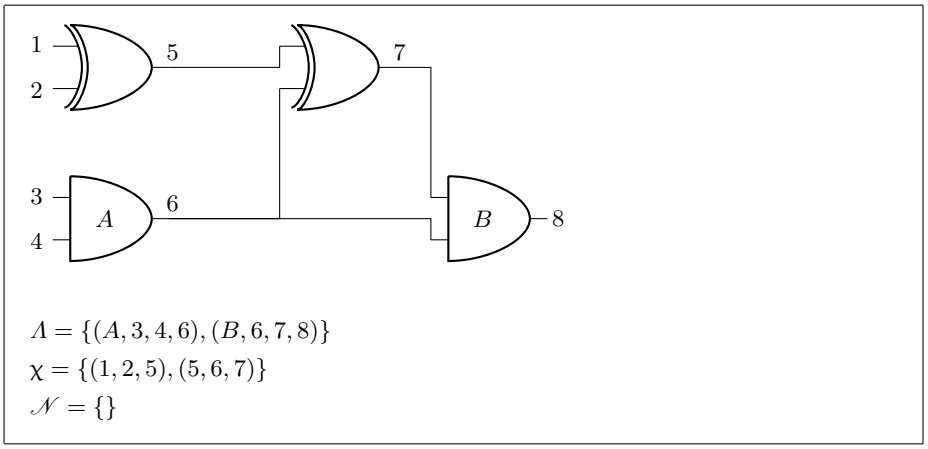

Fig. 8. Circuit representation example

7.2 The Protocol for circuit satisfiability

Our protocol for circuit satisfiability assumes 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, inversion and 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 $\Phi_{\mathtt{PLZK}}$. Pseudo-code is given below for an overview and concise description in Algorithm 1. Then Figure 9 depicts how the linear relations in S are proven. In full detail our protocol for circuit satisfiability goes as follows:

First, the verifier will commit to his challenge. Then both prover and verifier initialize a set S of pairs to be empty. This set will denote the set of linear relationships that will be tested.

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 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 each and gate. That is, the verifier flips a coin deciding whether to check if the associated helper triple 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. The prover will reveal the appropriate permutations and they will add the appropriate linear tests to verify the selected property for each and gate. We will then insert the linear tests for negation and xor relations into the set of things to be verified. The equality and inequality tests are added to the set S and all executed in

parallel at once when all permutations has been communicated for the entire circuit. Finally, the verifier and prover will use a modified version of $\Phi_{\texttt{PLZK}}$ where the challenge will be the value which the verifier committed to at the beginning of the protocol.

```
// commitment to the challenge
ε = com(E)
// Set of linear Relationships that
   will be proven
S ← ∅
// proof that circuit assignment has
   value 1
S ← S · ({n}, 1)
Prover
  Π, P ← ∅  // Permutation for
      triples and majority
  m ← {0}^{n+3a}
  for i ∈ {1, ..., n} do
      m_i ← v_i

   // generate triples, store
      permutations
  for (c, i, j, k) ∈ Λ do
      π ∈_R S_3
      Π ← Π · (c, π)
      (d_1, d_2, d_3) ← π(1, 2, 3)
      z ← n + 3(p − 1)
      m_{z+d_1} ← m_i
      m_{z+d_2} ← m_j
      m_{z+d_3} ← 0
      p ∈_R { μ ∈
      S_3 | μ(m_{z+1}, m_{z+2}, m_{z+3}) =
      (m_k, m_k, .) }
      P ← P · (c, p)

// commit to the string
Prover ↔ Verifier
  xom(w)
```

```
// select challenge for ands
Verifier
  b ∈_R {0, 1}
  Verifier --b--> Prover

// triples challenge
if b = 0 then
    Prover --Π--> Verifier
    for (c, i, j, k) ∈ Λ do
        d_1, d_2, d_3 ← π(1, 2, 3) where
        (p, π) ∈ Π
        z ← n + 3(p − 1)
        S ← S · ({i, z + d_1}, 0)
        S ← S · ({j, z + d_2}, 0)
        S ← S · ({z + d_3}, 0)
// majority challenge
if b = 1 then
    Prover --M--> Verifier
    for (c, i, j, k) ∈ Λ do
        d_1, d_2, d_3 ← m(1, 2, 3) where
        (p, m) ∈ M
        z ← n + 3(p − 1)
        S ← S · ({k, z + d_1}, 0)
        S ← S · ({k, z + d_2}, 0)

// insert xor relationships
for (i, j, k) ∈ χ do
    S ← S · ({i, j, k}, 0)

// insert inequality relationships
for (i, j) ∈ N do
    S ← S · ({i, j}, 1)
```

Algorithm. 1. Pseudo code describing how prover and verifier exchange permutations for and-gates and prepare the set, S, of linear relations to be proven.

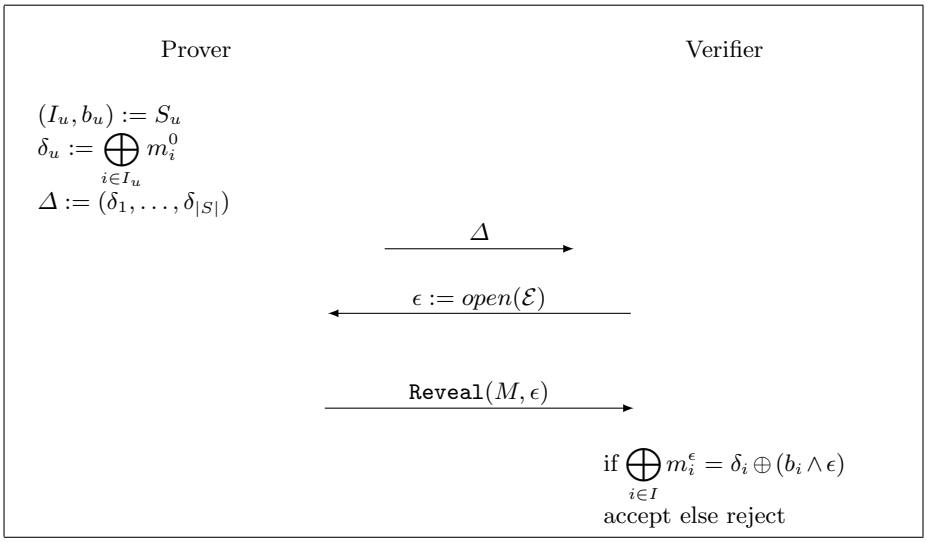

Fig. 9. Parallel proof for the relations added the set S by Algorithm 1

8 Weak Zero-knowledge

The weak Zero-knowledge functionality works as follows: on receiving an input and a witness from an honest prover, it verifies that the witness is valid for the given input and simply informs the verifier that a witness for the input has been proven. On the other hand, if the prover is corrupt, the weak Zero-knowledge functionality behaves differently.

Theorem 1. *$\Phi_{\texttt{WZK}}$ securely realizes $\mathrm{F}_{\texttt{WZK}}$ in the $\mathrm{F}_{\texttt{COM}}$ hybrid model.*

Prover corrupt The simulator plays the role of the verifier. First, he sends the message (`committed`) to the environment. The simulator awaits that the prover send the commit commands for the xor commitment. He verifies if the string committed to contains a valid assignment of the circuit. If so, he sends $(\texttt{witness}, x, w, \text{sid})$ to $\mathrm{F}_{\texttt{WZK}}$ where x is the circuit.

Otherwise, he sends $(\texttt{witness}, x, \bot, \text{sid})$ to $\mathrm{F}_{\texttt{WZK}}$, he checks which test for the and gates, would the simulator be able to always pass. If he could always pass the majority test he selects $a_1 = 0$, otherwise if he can always pass the triples test, he selects $a_1 = 1$ otherwise he selects $a_1 = \bot$. He then sends the command $(\texttt{perm}, a_1, \text{sid})$ and then receives $(\texttt{go}, a_2, \text{sid})$, it then forwards a_2 to the environment.

The simulator then awaits that the environment forwards the appropriate permutations. and the Δ of his choice. The simulator then determines for which challenge the test could be passed, if none then the simulator sends $(\texttt{test}, \bot, \text{sid})$ to the ideal functionality, if at least one then send $(\texttt{test}, b_1, \text{sid})$ where b is one of them to the ideal functionality. It then awaits to receive a $(\texttt{proceed}, b_2, \text{sid})$

$\mathcal{F}_{WZK}$

- On receipt of $(\texttt{witness}, x, w, sid)$ from the prover, there are four cases:
  1. If a command of the form $(\texttt{witness}, \cdot, \cdot, sid)$ has already been sent, ignore the command.
  2. If the prover has not been corrupted and $(x, w) \in R$ tell the verifier that $(\texttt{proven}, x, \text{sid})$.
  3. If the prover has not been corrupted and $(x, w) \notin R$, then ignore the command.
  4. If the prover has been corrupted and the command has not already been seen, record (x, w, sid).
- On receipt of $(\texttt{perm}, a_1, \text{sid})$ from the prover, if (x, w, sid) has not been recorded or a command of the form $(\texttt{perm}, \cdot, \text{sid})$ has already been sent, ignores the command. Otherwise select at random $a_2 \in \{0, 1\}$, it then records $(\texttt{go}, a_1, a_2, \text{sid})$ and sends $(\texttt{go}, a_2, \text{sid})$ to the prover.
- On receipt of $(\texttt{test}, b_1, sid)$ from the prover, if $(\texttt{go}, a_2, \text{sid})$ has not been sent to the prover or a command of the form $(\texttt{test}, \cdot, sid)$ has already been sent, ignore the command. Otherwise select at random $b_2 \in \{0, 1\}$, it then records $(\texttt{proceed}, b_1, b_2, \text{sid})$ and sends $(\texttt{proceed}, b_2, \text{sid})$ to the prover.
- On input $(\texttt{proof}, sid)$ from the prover, if $(\texttt{proceed}, b_2, \text{sid})$ has not been sent to the prover or $(\texttt{proof}, sid)$ has already been sent, ignore the command. Otherwise if $a_1 = a_2$ or $b_1 = b_2$ or if $(x, w) \in R$ then output $(\texttt{proven}, x, \text{sid})$ to the verifier

Fig. 10. The Weak Zero Knowledge Functionality

from the ideal functionality. The simulator then forwards $(\texttt{reveal}, b)$. It awaits that the environment reveals the given commitment.

The sender checks if a verifier would accept, if so he sends $(\texttt{proof})$ otherwise he aborts. Note that acceptance can only happen if a valid witness was entered or if $a_1 = a_2$ or $b_1 = b_2$.

8.1 Prover simulation indistinguishability

The real world and ideal world are indistinguishable because the choices given for the test to choose and the string to determine are dictated by the ideal functionality. The simulator can see for which tests the environment would pass. As a result, the environment can only pass in the ideal world if it would pass in the real world. The remaining messages are duplicated from the environment. As a result, the real and ideal world are indistinguishable.

Verifier corrupt The simulator awaits that the environment send the command $(\texttt{commmit}, \epsilon)$. Now the simulator knows the challenge ahead of time. The

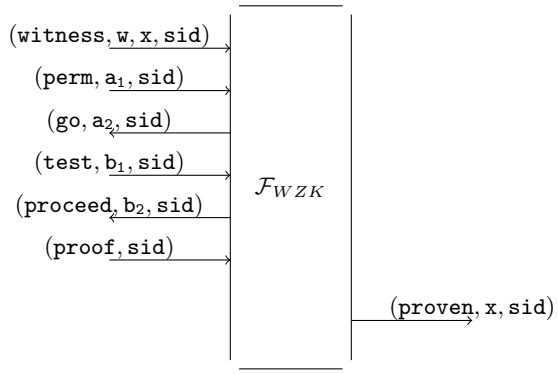

Fig. 11. Ideal functionality for Weak Zero-knowledge assuming a dishonest prover. The internal state of the functionality is hidden for readability.

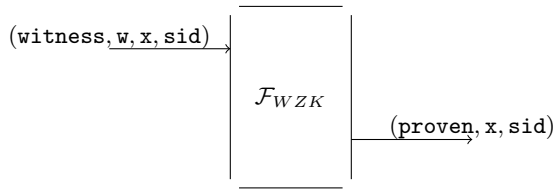

Fig. 12. Ideal functionality for Weak Zero-knowledge assuming an honest prover, again hidding the internal state for readability.

simulator then sends the commit message associated to the xor commitments, he also select random m^{ϵ} which he will decide to reveal. He now awaits the choice of test for the and gates from the environment. He selects random permutation for the given test and Δ such that he will be able to pass the given committed challenge. He then awaits that the environment send the open command associated to the commitment to the challenge. Finally, he sends the message $(\texttt{reveal}, m^{\epsilon})$ to the environment.

8.2 Verifier simulation indistinguishability

Notice, that aside from the Δ, m^{ϵ} and the permutation , the other messages are the same in the ideal and in the real world. Finally, the real world distribution of distributed variables are all sampled uniformly or are masked with a randomly sampled variable. As a result, the real and ideal world are indistinguishable.

Theorem 2. $\mathrm{F}_{\mathtt{ZK}}$ *can be securely instantiated using* $5s$ *calls* $\mathrm{F}_{\mathtt{WZK}}$.

see appendix A for details.

9 Improvement

For the protocol defined above, in the random oracle model, the commitment can be instantiated using a hash function to make the protocol more efficient.

The xor commitment consists of two commitment, the first one is used to commit to a random string. To reduce the communication, we could modify the xor commitment scheme so that instead of committing to a full length string. We instead commit to a short seed and then expand it using a pseudo-random number generator. The second string would be the message xored with this sequence. This means that opening the first string becomes very short. Simply send the commitment of the seed. We could also do the same trick to reduce the amount of communication for the permutations. Commit to the triples permutation by committing to a short seed and then fix the other majority permutations in consequence.

The final trick is to notice that the Δ in the proof are given to the verifier and then the verifier checks that these are indeed the values that he has. Now, an equivalent way of doing this, would be for the prover to simply send a $Hash(\Delta)$ to the verifier and after getting the opening of the message he could just hash the values of Δ, he would compute and hashes it and then verifies that this matches what he would get.

10 Conclusions

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

Currently we are experimenting with implementing our scheme using SHA-512 as the underlying commitment scheme. The full version will include an appendix with a more comprehensive review of our empirical result. The preliminary tests looks promising: we have our xor-commitment scheme running between two standard consumer grade machines. In our experiment we are committing to strings encoding circuits of up to two million gates in less than 4 seconds. That is, we measure the time it takes for the prover to create the commitments on one machine until they are transmitted to the verifier on a second machine.

We have circuits for AES, SHA-1, and SHA-256 [3] which are all smaller than this. For AES we hope to show knowledge of plaintext, the key or both given a ciphertext. For SHA we hope to show knowledge of preimages in zero knowledge. All with very shorty running time.

[3] Kindly borrowed from http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/ with many thanks to Nigel Smart and his crew at Bristol.

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. 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.
4. 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/.
5. 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.
6. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing*, pages 186–208, 1989.
7. 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.
8. 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.
9. 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.
10. Alain Tapp. Cryptography based on oblivious transfer. In *Proceedings of Pragocrypt 96*. 1996.

A Zero-knowledge from weak zero-knowledge

A.1 ZK from WZK

$\mathrm{F}_{\mathtt{ZK}}$

– Upon receiving $(\mathtt{proof}, x, w, sid)$ from the Prover, if a command of the form $(\mathtt{proof}, \cdot, \cdot, sid)$ has already been sent or if $(x, w) \notin R$, it ignores the command. Otherwise, it sends $(\mathtt{proven}, x, sid)$ to the verifier.

Fig. 13. Zero Knowledge Functionality

$\Phi_{\mathtt{ZK}}$

Verifier selects s unused session ids $S := \{\mathrm{sid}_1, \ldots, \mathrm{sid}_s\}$ and sends them to the prover. For each session id, the prover sends $(\mathtt{witness}, \mathtt{x}, \mathtt{w}, \mathtt{sid})$ to $\mathrm{F}_{\mathtt{WZK}}$.
Verifier accepts after receiving for each $i \in I$, the message $(\mathtt{proven}, x, \mathrm{sid}_i)$ from $\mathrm{F}_{\mathtt{WZK}}$

Fig. 14. Protocol for Zero Knowledge

Theorem 3. *$\Phi_{\mathtt{ZK}}$ securely realizes $\mathrm{F}_{\mathtt{ZK}}$ in the $\mathrm{F}_{\mathtt{WZK}}$, $\mathrm{F}_{\mathtt{COM}}$ hybrid model.*

Prover simulation The simulator sends to the environment a set of unused indices $S := \{\mathrm{sid}_1, \ldots, \mathrm{sid}_s\}$. In parallel, for each sid_i, the simulator awaits $(\mathtt{witness}, x, w, \mathrm{sid}_i)$. The simulator on reception of such a message, if such a message with the given sid_i has not already been sent, it stores (x, w, sid_i). The simulator then awaits a message of the form $(\mathtt{perm}, a_1, \mathrm{sid}_i)$. It then selects a random $a_2 \in \{0, 1\}$ and forwards $(\mathtt{go}, a_2, \mathrm{sid}_i)$, It then awaits $(\mathtt{test}, b_1, \mathrm{sid}_i)$, selects at random a $b_2 \in \{0, 1\}$ and sends $(\mathtt{proceed}, b_2, \mathrm{sid}_i)$ back to the receiver. It then awaits $(\mathtt{proof}, \mathrm{sid}_i)$. After having awaiting $(\mathtt{proof}, \mathrm{sid})$ and received them he checks if for all sid that the values stored associate to that id are such that $a_1 = a_2$ or $b_1 = b_2$ or $(x, w) \in R$. If this is the case for all of them then the simulator looks for a valid witness among the witnesses sent. He then sends $(\mathtt{proof}, x, w, \mathrm{sid})$

Prover simulation The real and ideal world can be distinguished if and only if the corrupted prover can succeed in passing all the weak zero-knowledge proofs without inputting a witness. This can only happen with negligible probability.

Corrupt verifier On input $(\texttt{proven}, x, \text{sid})$ from the zero-knowledge functionality, The simulator await the set of ids $S := \{\text{sid}_1, \ldots, \text{sid}_s\}$ from the environment. It then send for each $i \in \{1, \ldots, s\}$ the message $(\texttt{proven}, x, sid_i)$ to the environment.

Verifier simulation Proof is trivial.

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 7. 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 indicies 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 $\mathit{\Pi} \leftarrow \mathit{\Pi} \cdot (c, \pi)$
- 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 $\Phi_{\text{LZK}}(W, \{(I_1, 0), (I_2, 0)\})$ 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

$M = \text{xom}(m) = (\text{com}(\text{m}_0), \text{com}(\text{m}_1))$ Prover inequality between two bits of the committed string.

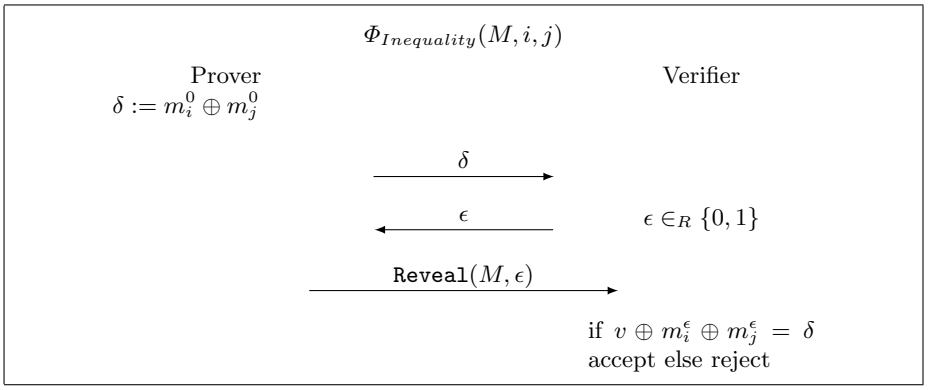

Fig. 15. Inequality

C.1 Zero-knowledge

Completness: 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 $M_{V^*}^{Neq}$

1 Select $b \in_R \{0, 1\}$
2 Compute $\delta \leftarrow b \oplus m_i^b \oplus m_j^b$
3 Accept to receive the $\epsilon \in \{0, 1\}$
4 If $\epsilon = b$ the simulator continues and opens the commitment to the message m^ϵ
5 Otherewise we rewind to step 1.

Fig. 16. Simulator for inequality

Clearly the distribution of δ send to the verifier is uniform as in the real execution. Also in Step 4 the simulator behaves exactly as in the real execution and the transcript exhibit the exact same distribution. □