Zero-Knowledge Using Garbled Circuits
or How To Prove Non-Algebraic Statements Efficiently

Marek Jawurek[1], Florian Kerschbaum[1], and Claudio Orlandi[2]

[1] SAP Research, Karlsruhe, Germany `marek@jawurek.net,florian.kerschbaum@sap.com`
[2] Aarhus University, Denmark `orlandi@cs.au.dk`

Abstract. Zero-knowledge protocols are one of the fundamental concepts in modern cryptography and have countless applications. However, after more than 30 years from their introduction, there are only very few languages (essentially those with a group structure) for which we can construct zero-knowledge protocols that are efficient enough to be used in practice.
In this paper we address the problem of how to construct efficient zero-knowledge protocols for generic languages and we propose a protocol based on Yao's garbled circuit technique.
The motivation for our work is that in many cryptographic applications it is useful to be able to prove efficiently statements of the form e.g., "I know x s.t. $y = \text{SHA-256}(x)$" for a common input y (or other "unstructured" languages), but no efficient protocols for this task are currently known.
It is clear that zero-knowledge is a subset of secure two-party computation (i.e., any protocol for generic secure computation can be used to do zero-knowledge). The main contribution of this paper is to construct an efficient protocol for the special case of secure two-party computation where only one party has input (like in the zero-knowledge case). The protocol achieves active security and is essentially only twice as slow as the passive secure version of Yao's garbled circuit protocol. This is a great improvement with respect to the *cut-n-choose* technique to make Yao's protocol actively secure, where the complexity grows linearly with the security parameter.

Table of Contents

1 Introduction

Zero-knowledge (ZK) protocols have countless applications in cryptography and therefore efficiency is of paramount importance. Consequently, a huge effort has been put into designing efficient ZK protocols for *specific tasks*. In particular there are very efficient protocols for languages with some algebraic structure. There are, for instance, efficient protocols for proving knowledge and relations of discrete logarithms [Sch89,CDS94], for proving that RSA public keys are well formed [CM99], for statements in post-quantum cryptography [BD10,JKPT12,MV03], for bilinear equations [GS12,GSW10], for shuffles [BG12,KMW12] and frameworks for modular design of zero-knowledge protocols [CKS11].

However, *generic constructions* for ZK protocols use Karp reductions to NP-complete languages and are therefore too impractical to be used in practice. In particular, so far there has been no practical solution to problems that do not exhibit an algebraic structure. Examples for protocols that could be used in many cryptographic applications are e.g., the problem of efficiently proving statements of the form "I know x s.t. $y = \text{SHA-256}(x)$" or "I know k s.t. $y_1 = \text{AES}_k(y_2)$" (the common input is y in the first example and (y_1, y_2) in the second)[3].

In this work we provide a *generic* and *efficient* solution for proving any such statements in zero-knowledge, by constructing a protocol based on Yao's garbled circuits technique. The complexity of our protocol is proportional to the size of the circuit of the NP verification function. To support the validity of our efficiency claim, we present also a *proof-of-concept* implementation of our protocol. The performance measurements of our prototypical implementation show the viability of our protocol for realistic problems.

Zero-Knowledge and 2PC. Zero-knowledge proofs were introduced more than 30 years ago by Goldwasser, Micali and Rackoff [GMR85]. A zero-knowledge argument (ZK) is an interactive protocol that allows a prover P to persuade a verifier V of the validity of some NP statement y by using the knowledge of a witness w. Informally, an honest prover should be able to convince an honest verifier of the validity of the statements (*completeness*). Moreover, a zero-knowledge protocol should give guarantees against corrupted parties: Even a malicious prover cannot persuade an honest verifier of a false statement (*soundness*) and even a malicious verifier does not learn anything from the execution of the protocol, except for the validity of the statement itself, hence the name "zero-knowledge".

Approximately at the same time, Yao [Yao82] introduced the problem of two-party secure computation (2PC) and showed how to solve it using the famous "garbled circuits" construction. In 2PC the parties hold secret inputs x_1 and x_2 respectively and want to jointly compute some function $z = f(x_1, x_2)$ while keeping their inputs secret.

Observe that ZK is a proper subset of 2PC: ZK is an instance of 2PC where there is an asymmetry between the parties and only P holds an input w (the witness for the NP statement) and where the function $f_y(w)$ outputs `accept` iff w is a valid witness for y (note that y here is not a secret, and therefore is not considered as an input but as part of the circuit description).

However, with few notable exception (see the related work in Section 1.2), the techniques used to implement efficient zero-knowledge protocols and the techniques used to implement efficient secure computation are very different from each other.

[3] Note that in both cases the prover is not only showing that the instance belongs to the language (both languages are trivial), but moreover that the prover *knows* a valid witness for this. So these proofs are meaningful as we believe that it is hard to compute such a witness.

ZK with 2PC techniques. From a very high-level point of view, our construction works as follows: V and P run the standard (passive-secure) Yao protocol, where V acts as the circuit constructor and P acts as the circuit evaluator. After the evaluation P sends V the output key (this step can be seen as a *conditional disclosure of secret* [GIKM98,AIR01]). The security of Yao's protocol implies that a computationally bounded P cannot guess the output key corresponding to the output value `accept` unless he has a valid witness for y.

This first, partial solution works in the case of a semi-honest verifier and it is described in Section 3. However, this protocol is not zero-knowledge against a malicious V, who can mount all the well-known active attacks against Yao's protocol (garbling a function different than the one he is supposed to, selective failure attack during the input phase [KS06] etc.).

Actively secure ZK. To make our protocol actively secure one can use off-the-shelf solutions for two-party computation with active security, like the *cut-n-choose* technique. In *cut-n-choose*, the circuit constructor sends multiple copies of the circuit to the circuit evaluator, who chooses a random subset of the circuits. Then the circuit constructor "opens" the chosen garbled circuits so that the other party can verify that they garble the correct function, and then they evaluate the remaining unopened garbled circuits. However even with the most recent versions of *cut-n-choose* (e.g. [FJN$^+$13,Lin13,HKE13]), the complexity of the resulting protocol still grows with the statistical security parameter and this represents a big drawback in practice.

Our approach for active security is different and takes advantage of the difference between ZK and 2PC. In the case of ZK the verifier V (i.e., the circuit constructor), has no secrets. Therefore, after the protocol is completed, V can "open" the circuit and let P verify that it computes the right function. This allows to achieve *active security* by using only a single garbled circuit.

1.1 Protocol idea.

In a nutshell our actively secure protocol works as follows: P and V run the OTs, where P uses as inputs the bits of his witness, and V inputs the keys for the input layer of the garbled circuit. Then V sends the garbled circuit (if the employed OT has two rounds, the garbled circuit can be sent with the second message of the OT). Now the prover can evaluate the circuit and retrieve the output key Z corresponding to the circuit output being "accept" or "reject". However, the prover does not reveal this key just yet. It rather commits to it and waits with revealing the actual value until after the verifier "opens" the garbled circuit and the prover can check the honesty of the verifier. Note that we need the OT protocol to have "committing" properties, as it is important for the prover to verify that the keys revealed in this stage are consistent with the ones used previously in the OT phase.

Applications. Private-key cryptographic primitives (such as block ciphers and hash functions) are orders of magnitude faster than their public-key counterparts: this is mostly due to the fact that the algebraic structure of many number-theoretic assumptions allows for more efficient attacks than those for symmetric primitives. Therefore to achieve the same level of security, the parameters used in public key cryptography are much bigger than the ones used in symmetric key cryptography.

However this extra algebraic structure allows for the construction of naturally efficient protocols and advanced functionalities. For example, AES encryption is much faster than RSA encryption. However, if one wants to be able to prove that the encrypted message has a particular structure (for

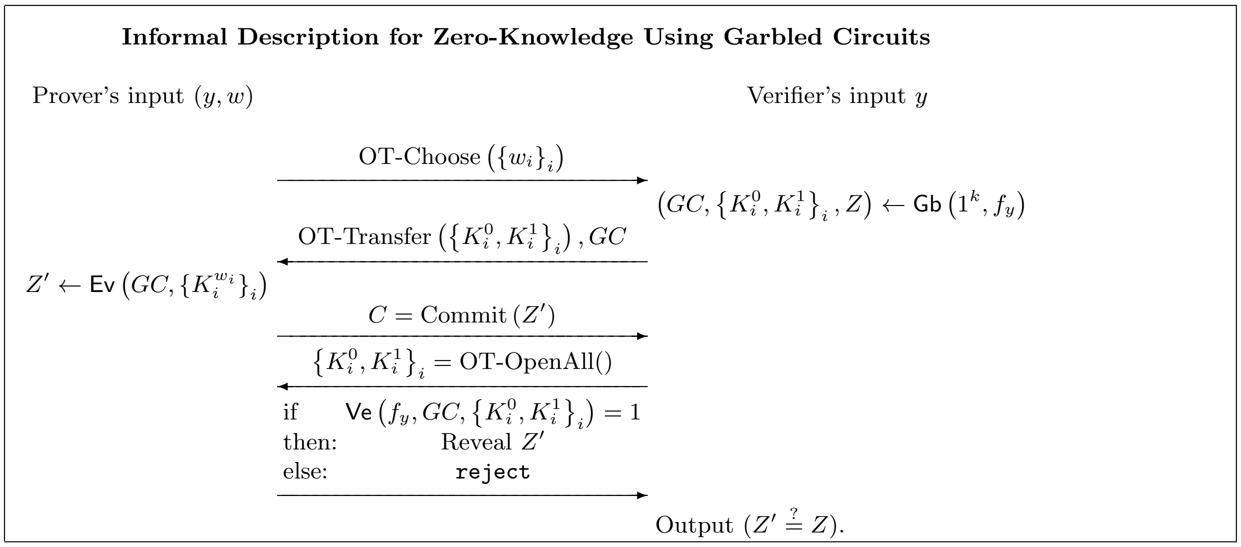
Informal Description for Zero-Knowledge Using Garbled Circuits
Prover's input (y, w)
Verifier's input y
OT-Choose $\left(\{w_i\}_i\right)$
$\left(GC, \left\{K_i^0, K_i^1\right\}_i, Z\right) \leftarrow \mathsf{Gb}\left(1^k, f_y\right)$
OT-Transfer $\left(\left\{K_i^0, K_i^1\right\}_i\right), GC$
$Z' \leftarrow \mathsf{Ev}\left(GC, \{K_i^{w_i}\}_i\right)$
$C = \text{Commit}\left(Z'\right)$
$\left\{K_i^0, K_i^1\right\}_i = \text{OT-OpenAll()}$
if $\mathsf{Ve}\left(f_y, GC, \left\{K_i^0, K_i^1\right\}_i\right) = 1$
then: Reveal Z'
else: reject
Output $(Z' \overset{?}{=} Z)$.

Fig. 1. An informal description of our zero-knowledge protocol.

instance, in escrow applications) one needs to take into account the efficiency of the zero-knowledge protocol as well, and therefore public-key primitives are sometimes favoured over symmetric key primitives because the public key schemes are compatible with the most efficient zero-knowledge protocols. The results of this paper change the state of things and allow for proving statements about cryptographic protocols with no algebraic structure in an practically efficient way.

As a motivating real life example, consider the following event: In 2010 Julian Assange released a "thermonuclear file insurance" i.e., a 1.4GB file allegedly consisting of an AES encryption of highly sensitive information. This encrypted data has been released as a countermeasure to protect WikiLeaks from being shut down by the U.S. government. The file had been extensively distributed using peer-to-peer application, and therefore simply releasing the encryption key would allow a great number of people to have access to this secret information. Note that it is important that the secret key is short, and can therefore be released in a very short time and with a wide variety of means, if one wants to.

It is natural to wonder whether this is just an empty threat (i.e., the file just contains random data) or whether the encrypted file really contains sensitive information. To show that the file actually contains sensitive information, one could imagine a randomized check in which the verifier asks to see the decryption of a random, but small portion of the file. To do this without releasing the decryption key a zero-knowledge protocol that "works well" with AES is required.

Finally, consider the Fiat-Shamir [FS86] heuristic for NIZK or signatures, widely used in practice due to its extreme efficiency. In many application it is desirable to prove that "I know a proof of x" or "I have a signature on m" without revealing the proof/signature. Doing so requires an efficient way to prove non-algebraic statement about the hash-functions used to instantiate the random oracle in the Fiat-Shamir construction.

1.2 Related Work

As discussed before, although zero-knowledge and secure computation are clearly related, few works have been exploiting MPC techniques in zero-knowledge protocols. Some few notable exceptions are listed here. Ishai, Kushilevitz, Ostrovsky and Sahai [IKOS07] use MPC protocols for honest majority to achieve efficient zero-knowledge protocols; their idea is to have the prover simulate "in its head" n parties running an MPC protocol for the verification function and commit to their internal states. Then the verifier gets to pick a subset of the parties and checks that their state is consistent with an accepting verification. Intuitively, given that the MPC protocol is secure against a minority of passively-corrupted players, the proof is zero-knowledge. This allows to construct zero-knowledge protocols with very good asymptotic complexity. However, as the authors goal is to optimize for asymptotic complexity, IKOS use tools that are rather cumbersome in practice (see Appendix D for a detailed comparison.). Designing a "practically efficient" protocol in the IKOS framework is an interesting open problem.

Gennaro, Gentry and Parno [GGP10] combine garbled circuits with fully-homomorphic encryption to obtain a verifiable computation protocol but due to the use of fully-homomorphic encryption this protocol is far from being practical. Bitansky and Paneth [BP12] also use Yao garbled circuits in order to achieve zero-knowledge protocols, however their constructions fall short of the goal and they only achieve *witness hiding* and *weak zero-knowledge* (two strictly weaker properties than zero-knowledge). Moreover, their construction uses point obfuscation as a tool, and therefore their security reduction makes non black-box use of the adversary and requires non-standard cryptographic assumptions. This is inherent, as their goal is to build 3-round protocols and it is known that black-box 3-round zero-knowledge is impossible under standard assumptions [GK96]. By relaxing the requirement on round-complexity we achieve full zero-knowledge using only standard assumptions and a black-box security reduction.

A number of papers investigate the feasibility (e.g., [GOS06a,GOS06b]) and the efficiency (e.g., [Gro10,Lip12,GGPR13]) of *non-interactive* zero-knowledge proofs for SAT. On the positive side, those protocols are non-interactive. On the negative side, all these protocols require expensive public key operations (exponentiations and pairings) for each gate in the circuit, and therefore our protocol is more efficient in practice, as it can be seen from the timings reported in [PHGR13]. It is a very interesting open problem to construct a protocol that is non-interactive and only uses cheap (symmetric) cryptographic primitives at the same time.

Structure. The remainder of this paper is structured as follows: Section 2 introduces and defines preliminaries for the presentation of our protocol. Section 3 then presents our protocol for an honest verifier as a warm-up. Then, in Section 4, we build upon that and present our actively secure protocol for a malicious verifier. Finally, in Section 5 we describe a prototype implementation and the results of our experiments regarding its performance.

2 Preliminaries

Let $[n] = \{1, \dots, n\}$. If S is a set $x \in_R S$ denotes a uniformly random element sampled from S. We call a function $\epsilon : \mathbb{N} \to \mathbb{R}^+$ negligible if for every polynomial p and big enough n, $\epsilon(n) < 1/p(n)$.

Let $L \subset \{0,1\}^*$ be a language in NP and M_L be the language verification function i.e., for all $y \in L$ there exist a string w of length polynomial in the size of y s.t. $M_L(y, w) = \texttt{accept}$ and for all $y \notin L, w \in \{0,1\}^*$ then $M_L(y, w) = \texttt{reject}$.

Defining Security. We use the standard security notion for static, active (malicious) adversaries defined as indistinguishability between the real world (where the adversary interacts with a honest party) and the ideal world (where the adversary interacts with a simulator with access to the ideal functionality), see [Can00,Can01,Gol04]. We will write our proofs in an hybrid model where real world parties have oracle access to a given functionality, while in the ideal world the simulator "controls" this functionality. We use the standard commitment functionality and slight augmentation of the oblivious transfer (OT) functionality.

All our simulators are straight-line and the protocols achieve UC-security [Can01] when instantiated with UC secure oblivious transfers, for instance [PVW08,DNO08].

For the sake of simplicity, we neglect some details from our ideal functionalities, such as the fact that all the communication can be deleted and delayed by the adversary. This has no impact on our proofs as we only prove static security.

Zero-Knowledge: The Ideal Functionality. We define the task of zero-knowledge with an ideal functionality, as in [Can01]. This is a convenient way of defining all the properties we want from a zero-knowledge protocol (including the *proof-of-knowledge* property) in a compact way.

The ideal functionality $\mathcal{F}_{\text{ZK}}^{R}$

- $\mathcal{F}_{\text{ZK}}^{R}$ is parametrized by a binary relation R.
- On input $(\texttt{prove}, sid, x, w)$ from P and $(\texttt{verify}, sid, x')$ from V, if $x = x'$ and $R(x, w) = 1$ output $(\texttt{verified}, x)$ to V.

Fig. 2. The ideal functionality $\mathcal{F}_{\text{ZK}}^{R}$ for Zero-Knowledge

The ideal functionality $\mathcal{F}_{\text{COT}}$

Choose: On input $(\texttt{choose}, id, b)$ from the receiver, with $b \in \{0,1\}$, if no messages of the form $(\texttt{choose}, id, \cdot)$ is present in memory, store $(\texttt{choose}, id, b)$ and send $(\texttt{chosen}, id)$ to the sender.

Transfer: On input $(\texttt{transfer}, id, tid, m_0, m_1)$ from the sender, with $m_0, m_1 \in \{0,1\}^k$, if no messages of the form $(\texttt{transfer}, id, tid, \cdot, \cdot)$ is present in memory and a message of the form $(\texttt{chosen}, id, b)$ is stored, send $(\texttt{transferred}, id, tid, m_b)$ to the receiver.

Open-all: On input $(\texttt{open-all})$ from the sender, reveal all messages $(\texttt{transfer}, id, m_0, m_1)$ to the receiver. Then the functionality halts and does not reply to any further command.

Fig. 3. The ideal functionality $\mathcal{F}_{\text{COT}}$ for OT with Sender Verifiability

The ideal functionality $\mathcal{F}_{\text{COM}}$

Commit: On input $(\texttt{commit}, id, m)$ from the sender, with $m \in \{0,1\}^*$, if no messages of the form $(\texttt{commit}, id, \cdot)$ is present in memory, store $(\texttt{commit}, id, m)$ and send $(\texttt{committed}, id, |m|)$ to the receiver.

Reveal: On input $(\texttt{reveal}, id)$ from the sender, if a message of the form $(\texttt{commit}, id, m)$ is present in memory send $(\texttt{reveal}, id, m)$ to the receiver.

Fig. 4. The ideal functionality $\mathcal{F}_{\text{COM}}$ for commitment

A Weak Flavor of Committing Oblivious Transfer. One of the ingredients in our construction is a flavor of oblivious transfer, defined in Figure 3. Similar primitives appeared in literature under the name of *verifiable OT* [Cré89], *committed OT* [CvdGT95], *authenticated OT* [NNOB12] just to name a few. The main difference here is that we only need to have commitments to the sender's input, and not the receiver, like in the notion of *committing OT* [KS06]. In fact, our flavor of OT is even weaker than *committing OT*, as we do not need to have individual commitments to the messages. The only extra property we need is that after all the OTs have been performed, the sender can reveal all its input messages to the receiver and cannot lie about them. This can in general be achieved by letting the sender commit to a seed in the beginning of the protocol, and then run any secure OT protocol using the output of a pseudorandom generator on the seed as its random tape. Then the `open-all` phase can be implemented by simply letting the sender reveal the seed and all the messages.

In Appendix A we show how some "natural" (and efficient) OT protocols already satisfy this extra property and securely implement the $\mathcal{F}_{\mathrm{COT}}$ functionality.

In Appendix C we show how the OT functionality can be further relaxed without impacting the security of our protocol. We do not need the protocol to be "correct" against a malicious sender, as long as the receiver will detect this during the `open-all` phase. This allows to use slightly less secure (and potentially more efficient) OT protocols.

Commitment. We describe the protocol in an hybrid model where parties have access to the $\mathcal{F}_{\mathrm{COM}}$ functionality. The $\mathcal{F}_{\mathrm{COM}}$ functionality can be implemented in a black-box way given the $\mathcal{F}_{\mathrm{COT}}$ functionality[4], or it is also possible to implement directly UC commitment in an efficient way using e.g., the protocol in [Lin11]). The (standard) commitment functionality is presented for completeness in Figure 4.

2.1 Garbling Scheme Requirements

We follow and extend the definition of garbled circuits by Bellare et al. [BHR12b]. A *verifiable* projective garbling scheme is defined by a tuple $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}, \mathsf{Ve})$[5] such that:

- The garbled circuit generation function Gb is a randomized algorithm that on input a security parameter 1^k and the description of a Boolean function $f : \{0,1\}^n \to \{0,1\}$, with $n = \mathrm{poly}(k)$ and $|f| = \mathrm{poly}(k)$ outputs a triple of strings (GC, e, d).
- The algorithm ev allows to evaluate the function described by f i.e., $\mathsf{ev}(f, x) = f(x)$.
- The encoding function En is a deterministic function that uses e to map an input x to a *garbled input* X. In this paper we will only be concerned with *projective* schemes where $e = \left(\{K_i^0, K_i^1\}_{i\in[n]}\right)$ and the garbled input X is simply $\{K_i^{x_i}\}_{i\in[n]}$, and therefore we will not further use the En function in the paper.
- The evaluation function Ev is a deterministic function that, evaluates a garbled circuit GC on an encoded input X and produces an encoded output Z'.
- The decoding function De, using the string d, decodes the encoded output Z' into a plaintext output z. In this paper we are only interested in whether $z = 1$ (i.e., the NP relation accepts), therefore we let $d = Z'$ and $\mathsf{De}(d, Z)$ outputs $z = 1$ if $Z' \overset{?}{=} Z$ or $z = 0$ otherwise (we will drop the De notation from now on).

[4] The committer inputs her message at the choice bit in the OT, while the verifier inputs two random strings as the messages of the OT. To decommit, the received random message is revealed. To break the binding property one would have to guess the message that was *not* chose in the OT, thus breaking the security of the OT protocol.

[5] A *standard* garbling scheme as defined by Bellare et al. does not contain the Ve procedure.

– In addition to the standard algorithms, a *verifiable* garbled scheme has an extra procedure Ve that, on input garbled circuit GC, a description of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and the input encoding information $e = \{K_i^0, K_i^1\}_{i\in[n]}$ outputs `accept` or `reject`.

The correctness and security definitions that we are going to provide are very similar to those given by Bellare et al., but they are relaxed to be as loose as they can while being sufficient for our goal. The hope is that our relaxed definitions can be achieved by simpler or more efficient garbling schemes that do not achieve the definitions given by Bellare et al.

Definition 1 (Correctness). *Let $\mathcal{G}$ be a verifiable projective garbling scheme described as above. We say that $\mathcal{G}$ enjoys* correctness *if for all $n = \mathrm{poly}(k), f : \{0,1\}^n \to \{0,1\}$ and all $x \in \{0,1\}^n$ s.t. $f(x) = 1$ the following probability*

$$\Pr\left(\mathsf{Ev}(GC, \{K_i^{x_i}\}_{i\in[n]}) \neq Z : (GC, \{K_i^0, K_i^1\}_i, Z) \leftarrow \mathsf{Gb}(1^k, f)\right)$$

is negligible in k.

Intuitively, Definition 1 says that it is possible to recover the secret Z by evaluating a honestly generated circuit with input keys corresponding to a value x such that $f(x) = 1$. This is clearly implied by the standard correctness requirement that asks for correctness on all possible values of x.

Definition 2 (Soundness). *Let $\mathcal{G}$ be a garbling scheme described as above. We say that $\mathcal{G}$ enjoys* authenticity *if for all $n = \mathrm{poly}(k), f : \{0,1\}^n \to \{0,1\}$ and all pairs $x \in \{0,1\}^n$ s.t. $f(x) = 0$ and for all PPT $\mathcal{A}$, the following probability:*

$$\Pr\left(\mathcal{A}(f, x, GC, \{K_i^{x_i}\}_{i\in[n]}) = Z : (GC, \{K_i^0, K_i^1\}_i, Z) \leftarrow \mathsf{Gb}(1^k, f))\right)$$

is negligible in k.

Intuitively, Definition 2 (which is a more succinct way of describing the "authenticity" property of Bellare et al., when restricted to our setting) says that no malicious evaluator can extract the secret Z unless she has access to input keys corresponding to a value x such that $f(x) = 1$.

Definition 3 (Verifiability). *Let $\mathcal{G}$ be a garbling scheme described as above. We say that $\mathcal{G}$ enjoys* verifiability *if for all $n = \mathrm{poly}(k), f : \{0,1\}^n \to \{0,1\}$ and all $x, y \in \{0,1\}^n$ with $x \neq y$ and $f(x) = f(y) = 1$ and for all PPT $\mathcal{A}$ the probability:*

$$\Pr\left(\mathsf{Ev}(GC, \{K_i^{x_i}\}_{i\in[n]}) \neq \mathsf{Ev}(GC, \{K_i^{y_i}\}_{i\in[n]}) : \begin{array}{l}\mathsf{Ve}(f, GC, \{K_i^0, K_i^1\}_{i\in[n]}) = \texttt{accept} \\ (GC, \{K_i^0, K_i^1\}_{i\in[n]}) \leftarrow \mathcal{A}(1^k, f)\end{array}\right)$$

is negligible in k.

In addition, we require the existence of a expected polynomial time algorithm Ext *s.t., for all x satisfying $f(x) = 1$ the probability:*

$$\Pr\left(\mathrm{Ext}(GC, \{K_i^0, K_i^1\}_{i\in[n]}) \neq \mathsf{Ev}(GC, \{K_i^{x_i}\}_{i\in[n]}) : \begin{array}{l}\mathsf{Ve}(f, GC, \{K_i^0, K_i^1\}_{i\in[n]}) = \texttt{accept} \\ (GC, \{K_i^0, K_i^1\}_{i\in[n]}) \leftarrow \mathcal{A}(1^k, f)\end{array}\right)$$

is negligible in k.

Intuitively, Definition 3 says that even a malicious constructor cannot create circuits that are successfully verifiable (i.e., make Ve output $\texttt{accept}$) and at the same time can output different values as a function of the evaluator's input x, as long as $f(x) = 1$. Jumping ahead, this is going to guarantee that the verifier cannot distinguish between different witnesses used by the prover.

Moreover, we require that the input of the testing function is enough to extract the secret in polynomial time. Note that this is trivial when it is easy to find an x s.t., $f(x) = 1$ but non-trivial otherwise. Jumping ahead, this extra guarantee will enable our simulator to extract the secret Z from the input of the malicious verifier to the oblivious transfer protocol, and it is therefore crucial to prove the zero-knowledge protocol. Intuitively, this is because this requirement ensures that the verifier already knows the (unique) secret Z when he sends the garbled circuit to the prover, and therefore the verifier is not learning any information when he receives back the secret Z as the output of the circuit evaluation by the prover (given that the check passes).

It is natural at this point to ask if there exist garbling schemes satisfying this definition. In Appendix B we discuss the most efficient Yao garbling schemes and argue how they satisfy all these properties under reasonable assumptions. Correctness follows from the correctness of Yao's construction. Intuitively soundness follows from the privacy of Yao's protocol (the output keys are encrypted using a secure encryption scheme under the input keys). The last requirement is satisfied by any garbling scheme good enough to be used in a *cut-n-choose* context. Note that the extraction requirement is very natural in Yao's scheme because, having access to all input keys, one can "fully decrypt" every garbled gate iteratively and get access to the key corresponding to the output bit 1 even without computing an input x such that $f(x) = 1$.

3 Warm-up: Honest Verifier Zero-Knowledge

It is trivial to construct zero-knowledge proofs when the prover is semi-honest. However, it is in some cases interesting to consider a relaxation of zero-knowledge in which the verifier is semi-honest i.e., where the verifier follows the protocol correctly but then tries to extract additional information about the prover's witness from the transcript of the protocol. This notion can be found in the literature under the name of *honest-verifier zero-knowledge* or HVZK.

In this section we start by presenting an efficient protocol for this scenario. Note however, that our protocols are private coin zero-knowledge protocols and therefore standard transformations from HVZK to full ZK cannot be used[6].

Constructing HVZK protocols using garbled circuits is relatively easy, and the main idea is as follows: the verifier, acting as the circuit constructor, constructs a Yao circuit that evaluates the function f_y. This function outputs 1 if $R(y, w) = 1$ otherwise it outputs 0. Then the parties run $n = |w|$ OTs, where V acting as the sender inputs the keys corresponding to the input wires and P inputs the bits of its witness $(w_1, \ldots, w_n)$. Now V sends the garbled circuit to P,[7] who evaluates and sends V the output of the evaluation of the garbled circuit. The formal description of the

[6] In a public-coin zero-knowledge protocol, the verifier does not have any private randomness and is limited to sampling random challenges and sending them to the prover. In this case it is possible to transform an HVZK into a ZK by letting the verifier commits to his challenges ahead of time and/or by sampling them using a coin flip protocol. In addition, public coin ZK protocols can be turned into non-interactive zero-knowledge protocols (NIZK) using the Fiat-Shamir heuristic. This is not possible for our construction, as here the messages sent by the verifier are not simply its random choices.

[7] Note that this is the proper order of things. If the circuit is sent before the OTs then the garbling scheme needs to be secure against adaptive attacks at the price of reduced efficiency, see [BHR12a].

protocol is provided in Figure 5. Note that the algorithm Ve is not needed in this case. Note also that if the OT protocol has a two move form, then the whole protocol has only 3 moves (first *choose* from P to V, then *transfer* and GC from V to P and finally S' from P to V).

The protocol π_{HVZK}.

Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{Ev})$ be a garbling scheme. Let L be an NP language with $|w| < n = \mathrm{poly}(|y|)$ and $f_y : \{0,1\}^n \to \{0,1\}$ be the verification function that outputs 1 if w is a valid witness for y.
Both parties have input y and a security parameter 1^k. In addition, the prover P has input $w = (w_1, \dots, w_n)$.

1. For all $i \in [n]$, P sends $(\texttt{choose}, i, w_i)$ to $\mathcal{F}_{\mathrm{OT}}$;
2. $\mathcal{F}_{\mathrm{OT}}$ sends V messages $(\texttt{chosen}, i)$ (for all $i \in [n]$);
3. V runs $(GC, \{K_i^1, K_i^0\}_{i\in[n]}, Z) \leftarrow \mathsf{Gb}(1^k, f_y)$;
4. For all $i \in [n]$, V sends to $\mathcal{F}_{\mathrm{OT}}$ the input $(\texttt{transfer}, i, K_0^i, K_1^i)$;
5. $\mathcal{F}_{\mathrm{OT}}$ sends P messages $(\texttt{transferred}, i, K_i')$ (for all $i \in [n]$);
6. V sends GC to P;
7. P runs $Z' \leftarrow \mathsf{Ev}(GC, \{K_i'\}_{i\in[n]})$ and sends Z' to V;
8. V outputs $\texttt{accept}$ iff $Z = Z'$;

Fig. 5. The protocol for honest-verifier zero-knowledge in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model

Theorem 1. *Let $\mathcal{G}$ be a garbling scheme satisfying Definition 1 and 2. Then protocol π_{HVZK} in Figure 5 securely implements the zero knowledge functionality $\mathcal{F}_{\mathrm{ZK}}^R$ in the presence of an actively corrupted P or a passively corrupted V in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model.*

Proof. When playing against a corrupted P^*: the simulator S extracts P^*'s input $w^* = (w_1^*, \dots, w_n^*)$ from the first step by simulating the $\mathcal{F}_{\mathrm{OT}}$ functionality and the simulator continues the protocol like an honest verifier with the only exception that at the end, it outputs $\texttt{accept}$ iff w^* is a valid witness for y. We argue that the view of P^* in the real world and in a simulated execution are computationally close as follows: up to the last step the view of P^* is distributed identically in the two settings (the simulator acts as an honest verifier except for its final output), and due to Definition 2 the probability that the corrupted P^* manages to make the real verifier output $\texttt{accept}$ when he does not input a valid witness in the OT phase is negligible.

The simulator can produce the view of a passively corrupted V^* by simulating the messages of the $\mathcal{F}_{\mathrm{OT}}$ functionality in Step 2. In Step 7, the simulator chooses sets Z' to be Z, the output of Gb (instead of the output of Ev). Due to Definition 1 the two values are equal except with negligible probability.

4 Zero-Knowledge from Garbled Circuits

In this section we describe our final protocol, that achieves full zero-knowledge also in the presence of a malicious verifier. An intuitive explanation of the protocol has been provided in the introduction, and therefore we proceed to formally describe the protocol in Figure 6 and prove its security. Note that the steps $1-6$ are the same as in the protocol π_{HVZK}.

Theorem 2. *Let $\mathcal{G}$ be a garbling scheme satisfying Definition 1, 2 and 3. Then protocol π_{ZK} in Figure 5 securely implements the zero knowledge functionality $\mathcal{F}_{\mathrm{ZK}}^R$ in the presence of actively corrupted parties in the $(\mathcal{F}_{\mathrm{COT}}, \mathcal{F}_{\mathrm{COM}})$-hybrid model.*

The protocol π_{ZK}.

Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{Ev}, \mathsf{Ve})$ be a garbling scheme. Let L be an NP language with $|w| < n = \text{poly}(|y|)$ and $f_y : \{0,1\}^n \to \{0,1\}$ be the verification function that outputs 1 if w is a valid witness for y.
Both parties have input y and a security parameter 1^k. In addition, the prover P has input $w = (w_1, \ldots, w_n)$.

1. For all $i \in [n]$, P sends $(\texttt{choose}, i, w_i)$ to $\mathcal{F}_{\text{COT}}$;
2. $\mathcal{F}_{\text{COT}}$ sends V messages $(\texttt{chosen}, i)$ (for all $i \in [n]$);
3. V runs $(GC, \{K_i^1, K_i^0\}_{i\in[n]}, Z) \leftarrow \mathsf{Gb}(1^k, f_y)$;
4. For all $i \in [n]$, V sends to $\mathcal{F}_{\text{COT}}$ the input $(\texttt{transfer}, i, K_0^i, K_1^i)$;
5. $\mathcal{F}_{\text{COT}}$ sends P messages $(\texttt{transferred}, i, K_i')$ (for all $i \in [n]$);
6. V sends GC to P;
7. P runs $Z' \leftarrow \mathsf{Ev}(GC, \{K_i'\}_{i\in[n]})$; In case the function Ev aborts, set Z' to $\perp$;
8. P send $(\texttt{committ}, 1, Z')$ to $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{COM}}$ outputs $(\texttt{committed}, 1, |Z'|)$ to V;
9. V sends the message $(\texttt{open-all})$ to the $\mathcal{F}_{\text{COT}}$ functionality;
10. $\mathcal{F}_{\text{COT}}$ sends P, for all $i \in [n]$, the values $(\texttt{transfer}, i, K_0^i, K_1^i)$;
11. P runs $\mathsf{Ve}(GC, \{K_i^0, K_0^1\}_{i\in[n]})$, if the output is not $\texttt{accept}$, P terminates the protocol. Otherwise, if Ve outputs $\texttt{accept}$, P sends $(\texttt{reveal}, 1)$ to $\mathcal{F}_{\text{COM}}$;
12. When V receives $(\texttt{reveal}, 1, Z')$ from $\mathcal{F}_{\text{COM}}$, V outputs $\texttt{accept}$ if $Z' \overset{?}{=} Z$;

Fig. 6. The protocol for honest-verifier zero-knowledge in the $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{COM}})$-hybrid model

Proof. When playing against a corrupted P^*: the simulator S extracts P^*'s input $w^* = (w_1^*, \ldots, w_n^*)$ from the first step by simulating the $\mathcal{F}_{\text{COT}}$ functionality and the simulator continues the protocol like an honest verifier with the only exception that at the end, it outputs `accept` iff w^* is a valid witness for y. We argue that the view of P^* in the real world and in a simulated execution are computationally close as follows: up to the last step the view of P^* is distributed identically in the two settings (the simulator acts as an honest verifier except for its final output): as discussed, the simulator only outputs `accept` when P^* uses a valid witness in the OT phase, while a real verifier will accept if the value received from the prover in step 11 is a valid opening of the commitment and Z is consistent with the secret hidden in the garbled circuit. But this can only happen with negligible probability if w is s.t. $f(w) = 1$: for the sake of contradiction, assume that P^* can use w s.t. $f(w) = 0$ in the OT phase and yet input $Z' = Z$ to the ideal commitment functionality in Step 8, then P^* can be used in a straightforward way to break the soundness property of the garbled circuit as defined in Definition 2.[8]

When playing against a corrupted V^*: The simulator sends messages $(\texttt{chosen}, i)$ to V^* and extracts V^*'s input $\{(K^*)_i^0, (K^*)_i^1\}_{i\in[n]}$ from step 4 by simulating the $\mathcal{F}_{\text{COT}}$ functionality. It then receives GC in step 6, sends the `committed` message to V^*, and waits to receive the message $(\texttt{open-all})$ from V^*. Finally in step 12, the simulator sends $Z' \leftarrow \text{Ext}(GC, \{(K^*)_i^0, (K^*)_i^1\}_{i\in[n]})$ to V^* if $\mathsf{Ve}(GC, \{(K^*)_i^0, (K^*)_i^1\}_{i\in[n]})$ accepts or $Z' = \perp$ otherwise.

Note that the simulator sends the `committed` message even if it already knows that the Ve is going to fail. We argue that this simulated view of a malicious verifier is indistinguishable from the view of the interaction with a real prover: note that if the verifier "cheats" (i.e., sends keys and circuit that make Ve reject) then the simulator always commits to the value $\perp$ in step 8 while an honest prover might commit to a different value (i.e., the output of the evaluation function on the garbled circuit with a specific set of input keys). As we are using an ideal commitment scheme,

[8] This is where we need to use the fact that the commitment is "extractable", as it guarantees that the adversary P^* computed the value Z using only the set of keys received from the OT phase and not the extra information received during the `openall` phase.

this does not change the distribution of the view of the adversary: in this case in fact (both in the simulated view and in the real world) the protocol ends at step 10, and the verifier does not get to see the committed value. In the other case i.e., when the verifier behaves honestly and the simulator commits to the secret value Z, then the view of the adversary in the real world and in the simulated execution is computationally close: this is guaranteed by Definition 3 that states that the output of the circuit is unique and therefore independent from the actual witness used – therefore the value committed by the simulator and the value committed by an honest prover are the same except with negligible probability.

5 Experimental Results

To show that our protocol π_{ZK} can be used in practice we implemented it and measured its performance. We stress that the goal of this section is not to show the best possible implementation of our protocol, but to show that the overhead to get active security on top of passive security when using garbled circuits for the case of zero-knowledge (as opposed to general 2PC) is very limited.

5.1 Choosing the Building Blocks

We chose only publicly available tools to implement our protocol, instead of creating an ad-hoc implementation from scratch. While this negatively affects our performance, it allows anyone interested in verifying our results to do so with only minimal effort.

Garbled Circuits. For the implementation of Gb and Ev we chose FastGC [HEKM11]. FastGC implements the evaluation of garbled circuits with state-of-the-art techniques like free-XOR [KS08] or pipelining [HEKM11]. Furthermore we employed GCParser [EMZ12] that parses circuit descriptions in an intermediate language for use with FastGC. This allows us to use the circuits available at [ST12] with little additional manipulation.

Commitment Scheme. The commitment scheme used by the prover in step 8 can be implemented as $\text{Com}(m, r) = \text{SHA-256}(m||r)$. This is an UC commitment scheme in the random oracle model: the use of the random oracle model seems necessary here, as we know that UC commitments are impossible to realize in the plain model and we do not wish to use impractical setup assumptions like the common reference string.

Note however that one has to be very careful and implement the commitment scheme in such a way that the commitment scheme is "non-malleable" with respect to the garbling scheme: this means that seeing the garbled circuit GC, the input keys and the opening in step 9 should not help a corrupted prover create and correctly open a commitment C for the secret Z. To see why this is necessary, consider an (artificial) garbling scheme that includes a commitment C to Z in the description of the garbled circuit GC. Now a malicious prover can simply send back to V this commitment C, wait for V to perform `open-all` in step 9. At this point he can open the commitment and make V accept, completely breaking the *soundness* of the protocol. To avoid this, one needs to use "different random oracles" for the commitment and for the circuit generator – in practice one could e.g., use different prefixes for the two different functions or use completely different functions e.g., AES to garble the circuit and SHA in the commitment.

Oblivious Transfer. We implemented Naor-Pinkas [NPS99] OT (NPOT) using the SCAPI library [EFLL12].As SCAPI's backend driver we chose the MIRACL Crypto SDK[9] and used the group of points of the Koblitz 224 curve. We tested the use of OT-extension, but due to the low number of input bits in our applications (the witness size), we found that using OT-extension produced worse results than simply using one OT per input bit.

5.2 Experiments

We have measured performance in four experiments:

HVZK AES: P proves knowledge of private k so that $c = \text{AES}_k(x)$ for public plaintext x and public ciphertext c using protocol π_{HVZK}. AES key expansion for k is performed in the circuit. The bit lengths of the circuit's inputs are: $|k|_2 = |c|_2 = |x|_2 = 128$. This is essentially a passive secure evaluation of AES, and therefore gives us a "base case" to measure the performance of our protocol.

ZK AES: As HVZK AES but this time we run the complete π_{ZK} protocol.

MD5: P proves knowledge of private x so that $h = \text{MD5}(x)$ for public hash h using π_{ZK}. The circuit only performs one invocation of the MD5 compression function, x is padded outside the circuit. The bit lengths of the circuit's inputs are: $|x|_2 = 512, |h|_2 = 128$.

SHA-256: P proves knowledge of private x so that $h = \text{SHA-256}(x)$ for public hash h using π_{ZK}. The circuit only performs one invocation of the SHA-256 compression function, x is padded outside the circuit. The bit lengths of the circuit's inputs are: $|x|_2 = 512, |h|_2 = 256$

Implementation details: In all experiments both parties, prover and verifier, were executed on the same "Intel(R) Core(TM) i7-2600 CPU 3.40GHz" and 16GBytes of RAM and averaged over 100 runs. However, prover and verifier had their own JVM and communicated over (local) network sockets.[10]

Performance measurements: In Figure 1 we present our results. The individually measured operations are characterized next[11]:

Circuit Parsing (1): P and V both independently parse the GC's description in the intermediate language. Furthermore this also includes some initializations. Clearly the time for parsing can be amortized over multiple executions or could be precomputed once and for all.

Oblivious Transfers P and V run NPOT with the witness and the keys corresponding the input wires as inputs.

Circuit Generation/Evaluation: P and V evaluate the GC using the pipelining technique: Both traverse the circuit in parallel and V creates and sends a garbled table (GTT) whenever he and P encounter a gate. P decrypts the respective entry in the GTT. FastGC's use of the free-XOR technique only requires GTT for AND and OR gates. Finally, P commits to the GC's output.

[9] `https://certivox.com/solutions/miracl-crypto-sdk/`

[10] As shown in the table, the communication complexity of the protocol is just a few hundreds kilobytes, so moving the protocol to a LAN should not change the timings significantly.

[11] Note that those operations are quite different from the idealized version of π_{ZK}. The reason for this is that in the "idealized" protocol V generates the whole circuit and then sends it to P. This generates a lot of idle time for the parties, while to get more efficiency it is useful to have both parties work in parallel.

Circuit Parsing (2): During the circuit verification, the prover parses the circuit again. This is just an implementation issue due to the "black-box" use of FastGC but could clearly be optimized away.

Circuit verification: V sends P the seed it used for the random creation of wire labels of the circuit and the OT. Furthermore, it also sends its inputs to the circuit to P. P starts evaluating it in the role of V. P compares the output it creates (the GTTs) to the GTTs it received from V previously. P accepts if both are equal for the entire GC. In this case it opens the commitment and V verifies the opening.

Discussion of results: Our experiments show that thanks to our protocol we can prove complex non-algebraic statements in only a few seconds.

We believe that our main contribution can be seen by comparing the timings of HVZK AES and ZK AES. Our protocol achieves active security on top of passive security with a slowdown of less than 20% with respect to the passive secure version, much better than the factor 2 we would have predicted.

The absolute value of $1.7s$ for AES ZK shows the competitiveness of our approach versus the standard techniques for active secure two-party computation: the best solution based on Yao on standard hardware [SS11][12] reports a time of $192s$ for one AES evaluation. The GMW-based active secure protocol in [NNOB12] requires $64s$ per AES instance (the timings go down to $2.5s$ but only when running many instances in parallel).

We conjecture that all our timings could be reduced by a significant factor if we were willing to sacrifice generality for performance: to see why, compare the timings for HVZK AES with the timings for passive secure 2PC of AES reported in [HEKM11], performing almost the same task[13]: they have a timing of $0.2s$ versus our $1.4s$. We believe that the main reason for this (apart from running the code on different hardware)[14] is that the AES circuit is hard-coded in their Java program and thus they do not perform circuit parsing at runtime – this generates also less overhead during evaluation. Therefore we believe that an ad-hoc implementation of our protocol for a specific circuit could run much faster than our prototype.

Acknowledgement. We thank Yan Huang for his support with FastGC and his OT implementation, Stefan Tillich for his support with their circuits and Yael Ejgenberg for support for the SCAPI framework. Finally we want to thank Omkant Pandey for pointing out that our construction requires a non-malleable commitment.

[12] [KSS12] reports better timings, but they run their experiments on machines with more than 60000 cores.

[13] Their implementation performs the AES key expansion at the prover side – this is not a problem when the prover is semi-honest but cannot be done in our case.

[14] They also instantiate the NPOT in the multiplicative group of $\mathbb{Z}_p$ with $p = 1024$ while we use the Koblitz 224 curve which is considered more secure.

	AES Decrypt				MD5		SHA256	
	HVZK		ZK		ZK		ZK	
Operation	Prover	Verifier	Prover	Verifier	Prover	Verifier	Prover	Verifier
Circuit Parsing (1)	239 ± 3	243 ± 4	239 ± 3	247 ± 3	336 ± 5	343 ± 3	565 ± 6	585 ± 9
Oblivious Transfers	731 ± 4	766 ± 5	736 ± 4	772 ± 5	2078 ± 7	2115 ± 7	2081 ± 7	2118 ± 8
Circuit Generation/Evaluation	335 ± 3	298 ± 3	332 ± 3	296 ± 3	605 ± 8	602 ± 8	1701 ± 7	1700 ± 7
Circuit Parsing (2)			91 ± 1		82 ± 1		240 ± 1	
Circuit Verification			163 ± 1		410 ± 2		1059 ± 15	
Total time	1404 ± 6		1667 ± 6		3617 ± 11		5761 ± 20	
Data transferred	256256 ± 0		262022 ± 2		1069647 ± 3		3052635 ± 4	
P's input bits	128				512		512	
#Gates	6927				29211		91080	

Table 1. Execution times and confidence intervals in milliseconds with confidence level 99%, averaged over 100 runs, rounded to ms. #Gates is the sum of AND and OR gates. The *total* runtime is the time (in ms) between the establishment of the network connection and the point where V accepts/rejects. *Data transferred* is the number of bytes exchanged between P and V. *P's inputs* indicates the number of input bits for which OTs have to be executed. *#Gates* is the number of non-free gates in the circuits.

References

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2001.

[BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In *TCC*, pages 201–218, 2010.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT*, pages 263–280, 2012.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy*, pages 478–492, 2013.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *ASIACRYPT*, pages 134–153, 2012.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, pages 784–796, 2012.

[BP12] Nir Bitansky and Omer Paneth. Point obfuscation and 3-round zero-knowledge. In *TCC*, pages 190–208, 2012.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. FOCS, 2001. Updated version at `http://eprint.iacr.org/2000/067`.

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

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In *TCC*, pages 39–53, 2012.

[CKS11] Jan Camenisch, Stephan Krenn, and Victor Shoup. A framework for practical universally composable zero-knowledge protocols. In *ASIACRYPT*, pages 449–467, 2011.

[CM99] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *EUROCRYPT*, pages 107–122, 1999.

[Cré89] Claude Crépeau. Verifiable disclosure of secrets and applications (abstract). In *EUROCRYPT*, pages 150–154, 1989.

[CvdGT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In *CRYPTO*, pages 110–123, 1995.

[DNO08] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. Essentially optimal universally composable oblivious transfer. In *ICISC*, pages 318–335, 2008.

[EFLL12] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. Scapi: The secure computation application programming interface. Cryptology ePrint Archive, Report 2012/629, 2012. `http://crypto.biu.ac.il/scapi`.

[EMZ12] David Evans, William Melicher, and Samee Zahur. *Interpreter for Garbled Circuits Intermediate Language*, 2012. `http://mightbeevil.org/gcparser/`.

[FJN$^+$13] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. Minilego: Efficient secure two-party computation from general assumptions. In *EUROCRYPT*, 2013.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, pages 626–645, 2013.

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In Jeffrey Scott Vitter, editor, *STOC*, pages 151–160. ACM, 1998.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *STOC*, pages 291–304. ACM, 1985.

[Gol04] Oded Goldreich. *Foundations of Cryptography Volume 2, Basic Applications*. Cambridge University Press, 2004.

[GOS06a] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for nizk. In *CRYPTO*, pages 97–111, 2006.

[GOS06b] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *EUROCRYPT*, pages 339–358, 2006.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

[GS12] Jens Groth and Amit Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM J. Comput.*, 41(5):1193–1232, 2012.

[GSW10] Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. Groth-sahai proofs revisited. In *Public Key Cryptography*, pages 177–192, 2010.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[HKE13] Yan Huang, Jonathan Katz, and Dave Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO*, 2013.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *STOC*, pages 21–30. ACM, 2007.

[JKPT12] Abhishek Jain, Stephan Krenn, Krzysztof Pietrzak, and Aris Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In *ASIACRYPT*, pages 663–680, 2012.

[KMW12] Shahram Khazaei, Tal Moran, and Douglas Wikström. A mix-net from any cca2 secure cryptosystem. In *ASIACRYPT*, pages 607–625, 2012.

[KS06] Mehmet S. Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of yao's garbled circuit construction. In *In Proceedings of 27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP (2)*, pages 486–498, 2008.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Towards billion-gate secure computation with malicious adversaries. *IACR Cryptology ePrint Archive*, 2012:179, 2012.

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the ddh assumption. In *EUROCRYPT*, pages 446–466, 2011.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, 2013.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, pages 169–189, 2012.

[MV03] Daniele Micciancio and Salil P. Vadhan. Statistical zero-knowledge proofs with efficient provers: Lattice problems and more. In *CRYPTO*, pages 282–298, 2003.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700, 2012.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.

[SS11] Abhi Shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT*, pages 386–405, 2011.

[ST12] Nigel Smart and Stefan Tillich. Circuits of basic functions suitable for mpc and fhe, 2012. `http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/`.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.

A Protocols for $\mathcal{F}_{\mathbf{COT}}$

In this section we discuss the relationship between the $\mathcal{F}_{\mathrm{OT}}$ functionality and the $\mathcal{F}_{\mathrm{COT}}$ functionality. While it is clear that the $\mathcal{F}_{\mathrm{OT}}$ functionality cannot be used in a black box way to run $\mathcal{F}_{\mathrm{COT}}$, we argue that any protocol based on computational assumptions can be used to implement $\mathcal{F}_{\mathrm{COT}}$. (Oblivious transfer can be instantiated using non-computational assumptions, such as noisy channels, quantum cryptography etc. We do not expect these kind of protocols to work in our context.) This can be done by letting the sender commit to a random seed s at the beginning of the protocol and then use the output of a pseudorandom generator $PRG(s)$ as its random tape in the protocol. Then during the `open-all` phase the sender reveals the seed and the messages and the receiver can check that the messages are consistent.

In order to make the paper self-contained, we describe here two efficient OT protocols that can be used to instantiate our construction, and discuss how those protocols already commit the sender to his choice of messages.

OT with UC-Security. We consider the UC-secure OT protocol proposed in [PVW08], instantiated using discrete logarithm. The reason for this choice is that the PVW OT appears to be the most efficient protocol for UC-OT in the literature. Let $(\mathbb{G}, q, g)$ be the description of a group of prime order q generated by g where the decisional Diffie-Hellman assumption is believed to hold.

CRS: The parties have access to a random string $(g_0, h_0, g_1, h_1) \in \mathbb{G}^4$.
Choose: R chooses a random $\alpha \in_R \mathbb{Z}_q$, computes $g = (g_b)^\alpha$, $h = (h_b)^\alpha$ and sends (g, h) to S;
Transfer: The sender operates in the following way:

1. S chooses random $r_0, s_0, r_1, s_1 \in_R \mathbb{Z}_q^4$ and computes

$$u_0 = g_0^{r_0} h_0^{s_0}, v_0 = g^{r_0} h^{s_0}$$

and

$$u_1 = g_1^{r_1} h_1^{s_1}, v_1 = g^{r_1} h^{s_1}$$

2. S sends R the values (u_0, w_0) where $w_0 = v_0 \cdot m_0$, and (u_1, w_1) where $w_1 = v_1 \cdot m_1$;

Retrieve: R outputs: $m_b = w_b \cdot (u_b)^{-\alpha}$

In [PVW08] the authors prove that this protocol securely implement the $\mathcal{F}_{\mathrm{OT}}$ functionality in the CRS-hybrid model. We can easily augment this protocol with the following procedure:

Open-All: S sends R the randomness r_0, r_1, s_0, s_1. R checks that

$$u_0 = g_0^{r_0} h_0^{s_0} \text{ and } u_1 = g_1^{r_1} h_1^{s_1}$$

and aborts if not. Otherwise, compute

$$v_0 = g^{r_0} h^{s_0} \text{ and } v_1 = g^{r_1} h^{s_1}$$

and output

$$m_0 = w_0 \cdot v_0^{-1} \text{ and } m_1 = w_1 \cdot v_1^{-1}$$

The sender cannot lie about the messages without breaking the discrete logarithm assumption: the values u_0, u_1 can be seen as Pedersen commitments, and if the adversary can find two different openings then the adversary can be used to find the discrete logarithms of h_i in base g_i for some $i \in \{0, 1\}$.

Naor-Pinkas OT: A problem with PVW OT is that it requires a common-reference string. Naor and Pinkas [NPS99] proposed a very efficient OT protocol in the plain model. It does not satisfy full-simulatabilty as PVW, but it is appears a good choice in practice as it uses fewer exponentiations and does not require a common reference string. The protocol proceeds as follows:

Setup: The sender chooses a random $d \in_R \mathbb{G}$ and sends it to the receiver.;
Choose: The receiver chooses a random $\alpha \in_R \mathbb{Z}_q$, computes $h_b = g^\alpha$ and $h_{1-b} = d \cdot (h_b)^{-1}$, and sends h_0 to the sender.
Transfer The sender computes $h_1 = d \cdot (h_0)^{-1}$, and sends

$$(c_0, c_1, c_2) = (h_0^r \cdot m_0, h_1^r \cdot m_1, g^r)$$

to the receiver.
Retrieve: R outputs: $m_b = c_b \cdot (c_2)^{-\alpha}$

This protocol can be augmented in the following way:

Open-All: The sender reveals r to the receiver, who checks that $c_2 = g^r$ and in this case outputs

$$m_0 = c_0 \cdot (h_0)^{-r} \text{ and } m_1 = c_1 \cdot (h_1)^{-r}$$

In this case the transfer message is a perfectly binding commitment to m_0, m_1, therefore no adversary can send inconsistent values.

We remark that if we use this OT protocol in our construction the simulator cannot extract the witness (i.e., the choice bits) of the prover/receiver, and in order to argue that the protocol π_{ZK} is also a proof-of-knowledge we need to rely on some extra assumption on the garbling scheme or by rewriting the proof in the random oracle model.

Saving on Communication Complexity: in both protocols the "Open-All" procedure simply consists of having the sender reveal all of its internal coins to the receiver. In practice one could have the sender producing his random tape by stretching a short seed with a pseudorandom generator. Then the receiver can regenerate the random tape on his side and perform the same computation as described above. This helps saving in communication complexity.

B Which Garbling Scheme Can Be Used?

The protocol described in Section 4 is compatible with every garbling schemes that can be used in standard cut-n-choose protocols. In particular, it is possible to use very optimized garbling schemes. To make the paper self-contained, we describe in this section such a garbling scheme that combines the state of the art optimizations for Yao Gates i.e., free XOR [KS08], permutation bits [NPS99], garbled row-reduction [NPS99].

This means that to garble a gate 4 evaluations of encryption are needed, and a garbled gate consists of only 3 ciphertexts (therefore saving on communication complexity). The evaluation of the gate consists of a single decryption.

How to garble gates. For the sake of exposition we will only describe how to garble NAND gates – as those are complete for Boolean circuits. It is straightforward to see how to garble other non-linear gates (remember that XOR-gates will be "for free").

Free-XOR: When generating the circuit, the circuit constructor chooses a random $\varDelta \in \{0,1\}^k$ and then, for each wire in the circuit, it will choose the key K_0 corresponding to the bit 0 at random in $\{0,1\}^k$ and set $K_1 = K_0 \oplus \varDelta$ for each wire in the circuit.
This allows to evaluate XOR gates for free: let $L_0, L_1 \in \{0,1\}^k$ be the keys associated to the left input wire and R_0, R_1 corresponding to the right input wire and O_0, O_1 the keys associated to the output wire. Remember that $L_0 \oplus L_1 = R_0 \oplus R_1 = O_0 \oplus O_1 = \varDelta$. Then, if the circuit constructor sets $O_0 = L_0 \oplus R_0$ it is easy to see that the circuit evaluator can use two keys L_a, R_b (without knowing a, b) to compute $O_{a\oplus b}$ as follows:

$$L_a \oplus R_b = L_0 \oplus a \cdot \varDelta \oplus R_0 \oplus b \cdot \varDelta = (L_0 \oplus R_0) \oplus (a \oplus b) \cdot \varDelta = O_{a\oplus b}$$

Encryption scheme: Let L, R be the input keys for a gate with identifier id and O the output key. We define some key derivation function KDF and encrypt the output key under the input keys as follows:

$$C = \mathsf{KDF}(L, R, id) \oplus O$$

In our implementation we use the same KDF as FastGC i.e., we hash the input keys and the id using SHA-1.
In [BHKR13], is suggested to use a different KDF based on the assumption that AES with a random (constant) key behaves like a random permutation. This can lead to efficiency improvements, especially when using new CPUs where AES encryption is implemented directly as a machine instruction.
Note that, as with every other free-XOR based construction, we need to assume that the KDF satisfies some kind of related key-attack security [CKKZ12].

Point-and-permute: We call the least significant bit of every key the permutation bit and write $p_K = lsb(K)$ and we assume that $lsb(\varDelta) = 1$. (This is not a problem as in our application the prover will learn all input keys (and therefore $\varDelta$) and can therefore check it during Ve.)
Then, for each gate in the circuit Gb samples random $L, R \in \{0,1\}^k$ and computes:

1. Define

$$\begin{aligned}\rho^0 &= p_L \wedge p_R\\ \rho^1 &= p_L \wedge \overline{p_R}\\ \rho^2 &= \overline{p_L} \wedge p_R\\ \rho^3 &= \overline{p_L} \wedge \overline{p_R}\end{aligned}$$

2. Compute $O_0 = \mathsf{KDF}(L_{p_L}, R_{p_R}, id) \oplus \rho^0 \varDelta$;
3. Compute:

$$\begin{aligned}\alpha_1 &= O_{\rho^1} \oplus \mathsf{KDF}(L_{p_L}, R_{\overline{p_R}}, id)\\ \alpha_2 &= O_{\rho^2} \oplus \mathsf{KDF}(L_{\overline{p_L}}, R_{p_R}, id)\\ \alpha_3 &= O_{\rho^3} \oplus \mathsf{KDF}(L_{\overline{p_L}}, R_{\overline{p_R}}, id)\end{aligned}$$

4. Output $(id, \alpha_1, \alpha_2, \alpha_3)$

Evaluation: The function Ev on input $(id, \alpha_1, \alpha_2, \alpha_3), L', R'$ does:

1. Define $\alpha_0 = 0^k$.
2. Compute $p_{L'} = lsb(L')$ and $p_{R'} = lsb(R')$ and let $j = 2p_{L'} + p_{R'}$.
3. Compute $O' = \alpha_j \oplus \mathsf{KDF}(L', R', id)$;

Remember that the requirement for correctness is: Let $(gg, O_0) \leftarrow \mathsf{Gb}(id, L_0, R_0, \Delta)$, then for all $a, b \in \{0,1\}$

$$\mathsf{Ev}(gg, L_a, R_b) = O_{a \wedge b}$$

except with negligible probability over the choices of L_0, R_0, Δ and the random coins of Gb and Ev.

This is the case because, let $L' = L_a, R' = R_b$, then $lsb(L') = a \oplus p_L$ and $lsb(R') = b \oplus p_R$. Then by construction $\rho^j = a \wedge b$ and

$$\alpha_j = O_{\rho^j} \oplus \mathsf{KDF}(L_a, R_b, id)$$

and therefore

$$O' = O_{\rho^j} = O_{(p_L \oplus a) \wedge (p_R \oplus b)} \ .$$

C A Relaxed OT Functionality

It turns out that while our zero-knowledge protocol in Section 4 is certainly secure when instantiated using a $\mathcal{F}_{\mathrm{COT}}$ functionality, this might be a bit of an overkill. We therefore introduce a second, relaxed version of verifiable OT as described in Figure 7. This functionality is insecure in the sense that it allows the sender to make the receiver's output depend on the receiver choice bits. Note that as our functionality allows for multiple OTs, this is actually a relaxation of the previous functionality: while it is always possible for the sender to make the receiver output any function of its input bit in a *single* instance of OT, this is not the case when considering *multiple* instances. In particular, being able to make the output of the jth OT depending on the choice bits in the ith OT, $i \neq j$, is a strong insecurity for an OT protocol.

On the other hand, if a malicious sender chooses to mount this attack, the functionality will inform the receiver of the malicious behavior of the sender when the command `open-all` is invoked.

Therefore the functionality is sufficient for our needs in the zero-knowledge protocol. The reason for describing this relaxation is that by relaxing the security requirement it might be possible to implement $\mathcal{F}_{\mathrm{COT}}$ with more efficient protocols than $\mathcal{F}_{\mathrm{OT}}$.

The ideal functionality $\mathcal{F}^{-}_{\mathrm{COT}}$

Choose: Unchanged from $\mathcal{F}_{\mathrm{COT}}$.
Transfer: Unchanged from $\mathcal{F}_{\mathrm{COT}}$.
Illegal-Transfer: Let m be the number of (`chosen`, id) messages received by the sender so far and $(b_1, \ldots, b_m)$ the selection bits input so far by the receiver. The ideal functionality accepts messages of the form (`illegal-transfer`, id, tid, g) where $g : \{0,1\}^m \to \{0,1\}^k$, with g an efficiently computable function and outputs (`transfer`, $id, tid, g(b_1, \ldots, b_m)$) to the receiver.
Open-all: On input (`open-all`) from the sender: if the ideal functionality received any messages of the form (`illegal-transfer`, ...) output (`corrupted-sender`) to the receiver, otherwise reveal all messages (`transfer`, id, m_0, m_1) to the receiver.

Fig. 7. The ideal functionality $\mathcal{F}^{-}_{\mathrm{COT}}$ for OT with Sender Verifiability with relaxed security

Intuitively, we can directly plug this functionality in the protocol π_{ZK} and still argue that π_{ZK} is secure. The argument is essentially the same as in the proof of Theorem 2: if a corrupted verifier performs an `illegal-transfer` the simulator will know this and commit to the $\perp$ value in step 8 (in the same way as the simulator does in π_{ZK} when the Ve function outputs `reject`), while an honest prover might evaluate the circuit on a set of maliciously chosen keys and commit to some value that potentially leaks information about the witness. However, during the `open-all` phase the honest prover will detect the cheating attempt and therefore abort the protocol.

It is an open problem to find protocols for OT (or OT-extension) that securely implement $\mathcal{F}^{-}_{\text{COT}}$ but not $\mathcal{F}_{\text{OT}}$ and are more efficient than the best known protocols that securely implement $\mathcal{F}_{\text{OT}}$.

D Comparison with IKOS

Ishai, Kushilevitz, Ostrovsky and Sahai (IKOS) [IKOS07] proposed a framework for asymptotically efficient zero-knowledge proof systems based on secure multiparty computation with honest majority. Here we briefly compare our results to theirs and describe the limitations of their protocols as described in their article. We stress that we do not exclude that their framework could also be optimized for "concrete efficiency". Indeed we believe that this is an interesting research direction.

IKOS present three main results: 1) a "simple protocol" based on 1-private 3-player MPC with the same asymptotic complexity as ours: it communicates ~ 3.5 times more than ours (considering the circuit size as the dominating term), and the computation (comparing the number of symmetric crypto primitives) is a factor of ~ 4.5 higher; 2) a protocol "approaching the witness length": let n be the witness size, s the circuit size and k the security parameter, then this protocol only communicates $n \cdot \text{poly}(k, log(s))$ but the computation of both prover and verifier is exponential in the circuit depth (for problems like AES that have depth around 40, this is clearly not a practical solution); 3) A protocol linear in the circuit size: with communication complexity $O(s) + \text{poly}(k)$ in contrast to $3ks$ in our protocol. However the IKOS protocol employs algebraic-geometric codes, that lead to constants of order 10^2 inside the big O notation. Thus in practice, it is not much better than our solution, where k is the security parameter for a symmetric crypto primitive (e.g., AES) and therefore has the same order of magnitude. In addition, the computation is $O(s) \cdot \text{poly}(k)$ in IKOS and $3ks$ in our protocol.