Efficient Zero-Knowledge Proof of Algebraic and Non-Algebraic Statements with Applications to Privacy Preserving Credentials

Melissa Chase[1], Chaya Ganesh[2], and Payman Mohassel[3]

[1] Microsoft Research, USA
[2] Department of Computer Science, New York University
[3] Visa Research, USA

Abstract. Practical anonymous credential systems are generally built around sigma-protocol ZK proofs. This requires that credentials be based on specially formed signatures. Here we ask whether we can instead use a standard (say, RSA, or (EC)DSA) signature that includes formatting and hashing messages, as a credential, and still provide privacy. Existing techniques do not provide efficient solutions for proving knowledge of such a signature: On the one hand, ZK proofs based on garbled circuits (Jawurek et al. 2013) give efficient proofs for checking formatting of messages and evaluating hash functions. On the other hand they are expensive for checking algebraic relations such as RSA or discrete-log, which can be done efficiently with sigma protocols.

We design new constructions obtaining the best of both worlds: combining the efficiency of the garbled circuit approach for non-algebraic statements and that of sigma protocols for algebraic ones. We then discuss how to use these as building-blocks to construct privacy-preserving credential systems based on standard RSA and (EC)DSA signatures.

Other applications of our techniques include anonymous credentials with more complex policies, the ability to efficiently switch between commitments (and signatures) in different groups, and secure two-party computation on committed/signed inputs.

1 Introduction

Efficient proofs. Zero knowledge proofs [GMR85] provide an extremely powerful tool, which allows a prover to convince a verifier that a statement is true without revealing any further information. Moreover, it has been shown that every NP language has a zero knowledge proof system[GMW87], opening up the possibility for a vast range of privacy preserving applications. However, while this is true in theory, designing proof systems that are efficient enough to be used is significantly more challenging. In reality, we only have a few techniques for efficient proofs, and those only apply to a restricted set of languages.

Almost exclusively, these proof systems focus on proving algebraic statements, i.e. statements about discrete logarithms, roots, or polynomial relationships between values [Sch90,GQ88,CS97b,GS08]. The most common and most efficient of these systems fall into a class known as sigma protocols. Of course we could express any NP relation as a

combination of algebraic statements, for example by expressing the relation as a circuit, and expressing each gate as an algebraic relation between input and output wires. But if we were to take this approach to prove a statement using sigma protocols we would need several exponentiations per gate in the circuit. This becomes prohibitively expensive for large circuits (for example a circuit computing a cryptographic hash function or block cipher).[4]

Recently, [JKO13] introduced a new approach for proving statements phrased as boolean circuits, based on garbled circuits. Their construction has the advantage that it only requires a few symmetric key operations per gate, making it dramatically more efficient than a sigma-protocol-based solution for non-algebraic statements. This means that it is finally practical to prove statements about complex operations such as hash functions or block ciphers. For instance, zero knowledge proofs for an AES circuit or a SHA256 circuit can be done in miliseconds on standard PCs using state of the art implementations for garbled circuits. On the other hand, expressing many public key operations as a circuit is still extremely expensive. (Consider for example a circuit computing modular exponentiation on a cryptographic group - the result would be much larger than the circuit computing a hash function, and computing a garbled circuit for such a computation would be too expensive to be practical.)

Now we have two very different techniques for achieving zero knowledge proofs for algebraic and non-algebraic statements. But in some applications, one is interested in proving statements that combine the two. For example, what if we want an efficient protocol for proving knowledge of a DSA or RSA signature, whose verification requires computing both a hash function and several exponentiations?

The state of the art fails to take advantage of the best of both worlds and has to forgo the efficiency of one approach to obtain the other's. One might consider directly combining both protocols, but a naive solution would allow a cheating prover to use a different witness for the algebraic and non-algebraic components of the computation and produce a convincing proof for a statement for which there is no single valid witness. Thus, one of the basic challenges is to bind the values committed to in the sigma protocols to the prover's inputs in the GC-based zero knowledge proof, without having to perform expensive group operations (e.g. exponentiation) inside the garbled circuit, and without proving large-circuit statements using sigma protocols.

Anonymous Credentials. Here, we primarily focus on the case of anonymous credentials, introduced by Chaum [Cha86], although we believe our results will be applicable to many other privacy protocols. A credential system allows a user to obtain credentials from an organization and at some later point prove to a verifier (either the same organization or some other party) that she has been given appropriate credentials. More specifically, the user's credentials will contain a set of attributes, and the verifier will require that the user prove that the attributes in his credential satisfy some policy. We say the system is anonymous if this proof does not reveal anything beyond this fact.

[4] SNARKs [Gro10,GGPR13] allow for very efficient verification and short proofs, but have similar shortcomings in prover efficiency as the prover performs public-key operations proportional to the size of the arithmetic circuit representing the statement.

There have been several proposals for constructions of anonymous credential systems [CL01,CL04,BCKL08,Bra99,BL13]. In general, they all follow a similar approach: the credential is a signature from the organization on the user's attributes. To prove possession of valid credentials, the user will first commit to her attributes, then prove, in zero knowledge, knowledge of a signature on the committed attributes, and finally prove, again in zero knowledge, that the committed attributes satisfy the policy. To make these zero knowledge proofs efficient, most of the proposed credential systems are based on sigma protocols, which as described above give efficient proofs of knowledge for certain algebraic statements. This in turn means that the signatures used must be specially designed so that a sigma protocol can be used to prove knowledge of a signature on a committed message.[5]

But what if we want to base our credentials on a standard signature such as FDH-RSA or DSA which includes hashing the message? Or what if we want the user to be able to prove a statement about his attributes that is not easily expressible as an algebraic relation?

Our Results. We study the problem of combining proof systems for algebraic and non-algebraic statements, and obtain the following results.

- Given an algebraic commitment C, we propose two protocols for proving that C is a commitment to x such that $f(x) = 1$ where f is expressed as a boolean circuit. Both constructions have the desired property that the GC-based component is dominated by the cost of garbling f (i.e. not garbling expensive group operations), and the total number of public-key operations is independent of the size of f.

  More specifically, our first solution has public key operations proportional to the maximum bit length of the input ($|x|$), and symmetric-key operations proportional to the number of gates in f. The second has public-key operations proportional to the statistical security parameter s and symmetric-key operations proportional to the number of gates in $f + |x|s$.

  Existing solutions either require public-key operations proportional to the size of f, or need to garble circuits for expensive group operations such as exponentiations in large groups.
- Building directly on these protocols, we show how to implement a proof that one committed message is the hash of another, and a proof that two commitments in different groups commit to the same value.
- Finally, we show how we can combine all of these protocols to obtain an efficient proof of knowledge of a signature on a committed message for RSA-FDH[6], DSA, and EC-DSA signatures.

[5] Technically, [Bra99,BL13] work slightly differently in that the user and organization jointly compute the proof of knowledge of a signature as part of the credential issuance. However they still use a customized issuing protocol which would not be compatible with standardized signatures, and they use sigma protocols exactly as described here to prove that the committed attributes satisfy the policy.

[6] This easily extends to standardized variants of RSA like RSA-PSS.

Applications.

- **Anonymous Credentials based on RSA, DSA, EC-DSA signatures.** The most direct application in the context of anonymous credentials would be to use RSA, DSA, or EC-DSA signatures directly as credentials but still allow for privacy preserving presentation protocols. This would be slower than existing credential systems, but it would have the advantage that the issuer would not have to perform a complex protocol, but would only have to issue standardized signatures. It further enables interoperability with existing libraries and non-private credential applications. [7]

  Alternatively, we could construct a service which allows users to convert their non-private credentials (based on RSA/DSA/EC-DSA signatures) into traditional anonymous credentials (e.g. Idemix [ide10] or UProve [PZ13] tokens, or keyed-verification credentials[CMZ14]). Using our new protocol, the service could perform that conversion *without knowing the user's attributes*: the user would commit to his attributes, prove using our protocol that they have been signed, and then obtain from the service an anonymous credential encoding the same attributes. (All of these anonymous credential systems allow for issuing credentials on committed attributes.)
- **Anonymous Credentials with more general policies.** Even if we consider a system based on traditional anonymous credentials, we might use the $\Pi_{\mathsf{Com},f}$ protocol (which we will describe in section 3) to allow the user to prove that his attributes satisfy a more complicated policy. For example, he might want to release the hash of one of his attributes and prove that that has been done correctly, or prove that an attribute has been encrypted using a standard encryption scheme like RSA-OAEP.

  Our protocols could also be used to prove that a user's attributes fall in a given range, or to prove statements about comparisons between attributes. If the range of values possible for each attribute is small, we already have reasonably efficient solutions - the user can just commit to each bit of the value, and do a straightforward proof. However this becomes expensive when the range gets larger, in which case the most efficient known approach is based on integer commitments [FO97] and requires several exponentiations with an RSA modulus where the exponent is larger than the group order (e.g. a roughly 2000 bit exponentiation with a 2000 bit modulus for reasonable security parameters). Alternatively we can use our second scheme, which only requires a number of public-key operations linear in the security parameter (e.g. 60), and allows those operations to use much more efficient elliptic curve groups.

  We note that the independent and concurrent work of [KKL$^+$16] provides an alternative solution to the problem of anonymous credentials for general policies, using different techniques.

[7] Delignat-Lavaud et al [DLFKP16] achieve a similar result using SNARKs, but with very different tradeoffs: their approach results in much shorter, non-interactive proofs, but much more expensive proof generation. They also explore several applications in more detail; in some of these applications, those which allow for interactive proofs, our protocols could be used to achieve these different tradeoffs.

- **Converting between different commitment schemes.** There are many protocols based around commitments, and ideally we would be able to combine these protocols arbitrarily. For example, if we have an efficient protocol for proving that a committed tag matches one of the attributes in a user's credential, and another protocol for proving that a committed tag is not on a list of revoked values, then we would be able to combine the two protocols to prove that the user's credential has not been revoked. However, often the protocols will be based on different commitment schemes, or even worse, on schemes that operate in different sized groups. (For example UProve credentials can be instantiated in standardized elliptic curve groups like those used for EC-DSA, while revocation systems like that in [Ngu05] require pairing groups; to combine the two we would need to find a pairing group whose group order matches one of the standardized curves. Finding a pairing group to match a specific group order often incurs a significant cost in efficiency.) With our protocol for converting between commitment schemes we could choose the most efficient groups for each, and then the user would merely prove that he has used the same attributes in each. Before our work, the only known approach to convert between groups of different sizes was to use integer commitments, which as described above can be quite expensive.
- **Other privacy-preserving protocols.** We note that while anonymous credentials make a good motivating application, these problems (converting between commitments schemes, comparing committed values, or proving other non-algebraic statements) come up in many other privacy/anonymity scenarios.
- **2PC with authenticated input.** As input to a secure computation protocol, sometimes it is desirable to use previously committed [JS07] or signed [CZ09] inputs. In our constructions, we show how to commit to an input x and prove knowledge of x (or prove knowledge of a signature on x) and a non-algebraic statement $f(x) = 1$ using garbled circuits. As we discuss in section 3.4, it is relatively easy to extend our construction to also allow secure two-party computation of $g(x, y)$ where x is the prover's input and y the verifier's, hence obtaining secure two-party computation on signed/committed inputs. The benefit of this approach is that checking the signature takes place outside the secure two-party computation and can be significantly more efficient.

2 Preliminaries

2.1 Simulation-based Security

We use a simulation-based definition of security in the ideal/real world paradigm, which is formulated by specifying an ideal functionality. A protocol is secure if it "emulates" this ideal functionality in the presence of any adversary. Our definitions are in the standalone setting (as opposed to the UC framework). We formulate the simulation-based definitions by defining a functionality $\mathcal{F}$ in the ideal world. In the ideal world, all parties and the adversary $\mathcal{A}$ interact via $\mathcal{F}$. Let $IDEAL_{\mathcal{F},\mathcal{A}}(x_1, x_2)$ denote the output vector of the adversary and the honest party from the execution in the ideal world. In the real world, a protocol π is executed among the parties, and let $REAL_{\pi,\mathcal{A}}(x_1, x_2)$ denote the output of the adversary and the honest party from the execution of π. A two party

protocol π securely realizes the functionality $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ in the real world, there exists a PPT adversary $\mathcal{S}$ in the ideal-world, such that

$$\{IDEAL_{\mathcal{F},\mathcal{S}}(x_1,x_2)\}_{x_1,x_2 s.t |x_1|=|x_2|} \stackrel{c}{\equiv} \{REAL_{\pi,\mathcal{A}}(x_1,x_2)\}_{x_1,x_2 s.t |x_1|=|x_2|}$$

that is, the two distributions are computationally indistinguishable.

2.2 Commitment Scheme

A commitment protocol involves two parties: the committer and the receiver. At a high level, it consists of two stages, a commitment phase and a de-commitment phase. In the commitment stage, the committer with a secret input m engages in a protocol with the receiver. At the end of this protocol, receiver does not know what m is (hiding property), and at the same time, the committer, can subsequently in the de-commitment phase, open only one possible value of m (binding property). Throughout the paper, we use algebraic commitment schemes that allow proving linear relationships among committed values. An example of such a scheme with computational binding and unconditional hiding properties based on the discrete logarithm problem is the one due to Pedersen [Ped91]. It works in a group G of prime order q. Given two random generators g and h such that $\log_g h$ is unknown, a value $x \in \mathbb{Z}_q$ is committed to by choosing r randomly from $\mathbb{Z}_q$, and computing $C_x = g^x h^r$. Protocols are known in literature to prove knowledge of a committed value, equality of two committed values, and so on, and the protocols can be combined in natural ways. In particular, Pedersen commitments allows proving linear relationships among committed values: Given C_x and C_y, prove that $y = ax + b$ for some public values a and b.

2.3 Committing OT

Similar to [JKO13] we need to need an OT protocol with a sender verifiability property- i.e. that at the end of the OTs, the sender is committed to its messages, and can be asked to reveal all its input messages to the receiver. This is closely related to the notion of committing OT [KS06], but can be achieved even more generally since we do not require individual commitments to sender's messages. In particular, as discussed in [JKO13] it can be satisfied by a protocol where the sender commits to a seed in the beginning of the protocol, and then runs any secure OT protocol using the output of a pseudorandom generator on the seed as its random tape. Then the open phase can be realized by letting the sender reveal the seed and all the input messages. The ideal functionality $\mathcal{F}_{COT}$ is defined in Figure 1.

Fig. 1. The ideal functionality $\mathcal{F}_{COT}$

- The receiver inputs $(choose, b), b \in \{0,1\}$, and the sender inputs (m_0, m_1).
- Output m_b to the receiver.
- On input open from the sender, send (m_0, m_1) to the receiver.

2.4 Garbled Circuits

We assume some familiarity with standard constructions of garbled circuits. We employ the abstraction of garbling schemes [BHR12] introduced by Bellare et al., but similar to [JKO13] we add a verification algorithm that can check correctness of the garbled circuit given all input labels to the circuit.

A garbling scheme is defined by a tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Eval}, \mathsf{Ve})$ such that:

- Gb is a randomized garbled circuit generation function that takes a security parameter, and the description of a boolean circuit f and outputs a garbled circuit GC and the encoding and decoding information e and d, respectively.
- The En algorithm takes the encoding information e output by Gb, and an input x to f, and outputs the garbled input corresponding to x.
- The Eval algorithm takes the garbled circuit GC and the garbled input, and outputs an encoded output.
- The De algorithm gets the encoded output and the decoding information d as input and returns a decoded output.
- The Ve algorithm gets as input a garbled circuit GC, the encoding information e, and a boolean function f, and outputs d or $\perp$.

In our constructions, we assume that the encoding information e is a vector of pairs of input labels, where the pair (K_i^0, K_i^1) denotes the input labels for 0 and 1 for input wire i in the circuit. Similarly, we assume that the decoding information d is a vector of pairs of output labels.

A garbling scheme may satisfy several properties such as *correctness, authenticity and privacy*. We review these notions next.

Definition 1. *A garbling scheme satisfies* **correctness** *if:*

- *for all boolean circuits f and all input x,*

$$\mathsf{De}(d, \mathsf{Eval}(GC, \mathsf{En}(e, x))) = f(x) \text{ whenever } (GC, e, d) \leftarrow \mathsf{Gb}(f, 1^\kappa)$$

- *for all boolean circuits f and all (possibly malicious) garbled circuits GC and encoding information e, decoding information d, and all input x,*

$$\text{if } d \leftarrow \mathsf{Ve}(GC, e, f) \text{ and } d \neq \perp \text{ then } \mathsf{De}(d, \mathsf{Eval}(GC, \mathsf{En}(e, x))) = f(x)$$

Definition 2. *A garbling scheme has* **authenticity** *if for every circuit f, input x, and PPT algorithm $\mathcal{A}$, the following probability*

$$\Pr[\exists y \neq f(x), y = \mathsf{De}(d, d') : (GC, e, d) \leftarrow \mathsf{Gb}(f, 1^\kappa), d' \leftarrow \mathcal{A}(GC, \mathsf{En}(e, x))]$$

is negligible in κ.

Definition 3. *A garbling scheme has* **privacy** *if there exists a PPT simulator $\mathcal{S}$ such that the following two distributions are indistinguishable:*

- $Real(f, x)$: *run* $(GC, e, d) \leftarrow \mathsf{Gb}(f, 1^\kappa)$*, and output* $(GC, \mathsf{En}(e, x), d)$.
- $Ideal_{\mathcal{S}}(f, f(x))$*: output* $\mathcal{S}(f, f(x))$

2.5 Zero-knowledge Proofs

A Zero-knowledge (ZK) proof allows a prover to convince a verifier of the validity of a statement, without revealing any other information. Let $\mathcal{L}$ be the language associated with an NP relation R: $\mathcal{L} = \{x \mid \exists w : R(x, w) = 1\}$. A zero-knowledge proof for $\mathcal{L}$ lets the prover convince a verifier that $x \in \mathcal{L}$ for a common input x. A proof of knowledge captures not only the truth of a statement $x \in \mathcal{L}$, but also that the prover "possesses" a witness w to this fact. A proof of knowledge for a relation $R(\cdot, \cdot)$ is an interactive protocol where a prover P convinces a verifier V that P knows a w such that $R(x, y) = 1$, where x is a common input to P and V. The prover can always successfully convince the verifier if indeed P knows such a w. Conversely, if P can convince the verifier with reasonably high probability, then it "knows" such a w, that is, such a w can be efficiently computed given x and the code of P. The formal definition follows. In the following, $view_V$ is the "view" of the verifier in the interaction, consisting of its input x, its random coins, and the sequence of the prover's messages.

Definition 4 (ZK proof of knowledge). *An interactive protocol $\langle P, V \rangle$ is a zero-knowledge proof of knowledge for an NP relation R if the following properties are satisfied.*

1. *Completeness: $\forall x, y$ such that $R(x, y) = 1$,*

$$\Pr[\langle P(x, w), V(x) \rangle = 1] = 1$$

2. *Proof of Knowledge: For every polynomial time prover strategy $P^*, \exists$ an oracle PPT machine K called the extractor such that $K^{P^*}(x)$ outputs w' and*

$$\Pr[\langle P^*(x, w), V(x) \rangle = 1 \wedge R(x, w') = 0]$$

   is negligible in κ.
3. *Zero-knowledge: For every polynomial time verifier V^*, there is a PPT algorithm $\mathcal{S}$ called the simulator such that for every $x \in \mathcal{L}$, the following two distributions are indistinguishable:*
   - $view_{V^*}(\langle P(x, w), V^*(x) \rangle)$
   - $\mathcal{S}(x)$

Honest-verifier zero-knowledge: An interactive proof system (P, V) for a language $\mathcal{L}$ is said to be honest-verifier zero knowledge if there exists a PPT algorithm $\mathcal{S}$ called the simulator such that for all $x \in \mathcal{L}$, $view_V(\langle P(x, w), V(x) \rangle)$ and $\mathcal{S}(x)$ are indistinguishable. This definition says that the verifier gains no knowledge from the interaction, as long as it runs the prescribed algorithm V. If the verifier tries to gain some knowledge from its interaction with the prover by deviating from the prescribed protocol, we should consider an arbitrary (but efficient) cheating verifier V^* as in the property 3 of the above definition which is full zero-knowledge. Efficient zero knowledge proofs are known which are based on sigma protocols. Sigma protocols are three round public-coin protocols and are honest-verifier zero-knowledge proof systems. There exist sigma protocols for various tasks like proving knowledge of discrete logarithm of a value, that a tuple is of the Diffie-Hellman type etc., and it is also possible to efficiently combine sigma protocols to prove compound statements. It is

possible to efficiently compile a sigma protocol (which is honest-verifier ZK) into a zero-knowledge proof of knowledge. The Fiat-Shamir transform [FS86] converts any public-coin zero-knowledge proof into a zero-knowledge proof of knowledge and removes interaction, and is secure in the random oracle model [PS96]. Transformations in the common reference string model [Dam00,Lin15] are also known. The transformation of [Dam00] gives a 3-round concurrent zero-knowledge protocol in the CRS model, whereas [Lin15] is non-interactive.

In our constructions and protocols, we make use of interactive zero knowledge proofs of knowledge of discrete logarithms and relations between discrete logarithms. We use the following notation:

$$\mathsf{PK}\{(x, y, \cdots) : \textit{ statements} \text{ about } x, y, \cdots\}$$

In the above, $x, y, \cdots$ are secrets (discrete logarithms), the prover asserts knowledge of $x, y, \cdots$, and that they satisfy *statements*. The other values in the protocol are public.

2.6 ZK Proof Based on Garbled Circuits

Here, we review an important building block for our construction, i.e., the garbled-circuit-based ZK protocol of [JKO13]. To prove a statement $\exists w : R(x, w) = 1$ (for public R and x), the protocol proceeds as follows:

1. The verifier generates a garbled circuit computing $R(x, \cdot)$. Using a committing oblivious transfer, the prover obtains the wire labels corresponding to his private input w. Then the verifier sends the garbled circuit to the prover.
2. The prover evaluates the garbled circuit, obtaining a single garbled output (wire label). He commits to this garbled output.
3. The verifier opens his inputs to the committing oblivious transfer, giving the prover all garbled inputs. From this, the prover can check whether the garbled circuit was generated correctly. If so, the prover opens his commitment to the garbled output; if not, the prover aborts.
4. The verifier accepts the proof if the prover's commitment holds the output wire label corresponding to TRUE.

Security against a cheating prover follows from the properties of the circuit garbling scheme. Namely, the prover commits to the output wire label before the circuit is opened, so the *authenticity* property of the garbling scheme ensures that he cannot predict the TRUE output wire label unless he knows a w with $R(x, w) = \text{TRUE}$. Security against a cheating verifier follows from correctness of the garbling scheme. The garbled output of a *correctly generated* garbled circuit reveals only the output of the (plain) circuit, and this garbled output is not revealed until the garbled circuit was shown to be correctly generated.

Note that in this protocol, the prover evaluates the garbled circuit on an input which is completely known to him. This is the main reason that the garbled circuit used for evaluation can also be later opened and checked for correctness, unlike in the setting of cut-and-choose for general 2PC. Along the same lines, it was further pointed out in [FNO15] that the circuit garbling scheme need not satisfy the *privacy* requirement of

[BHR12], only the *authenticity* requirement. Removing the privacy requirement from the garbling scheme leads to a non-trivial reduction in garbled circuit size.

In one of our constructions (section 3.2), the verifier does have a private input, but its input only needs to be kept private until the circuit is evaluated and the prover has committed to the output. In that scenario, we also invoke the *privacy* property of the garbling scheme as defined above.

Efficiency of Garbling Schemes. The state of the art garbling scheme uses the free-XOR technique [KS08] to garble XOR gates and the half-gate technique to garble AND gates [ZRE15]. For a circuit with g non-XOR gates, the total number of ciphertexts is $2g$, and the number of hash invocations is $4g$ for the garbler and $2g$ for the evaluator.

For *privacy-free* garbling, the costs are reduced by factor of two (see [FNO15,ZRE15]). In particular, for a circuit with g non-XOR gates, the total number of ciphertexts is g, and the number of hash invocations is $2g$ for the garbler and g for the evaluator.

We need to garble a few common building-block circuits in our constructions. It is helpful to review the size of these circuits based on the concrete constructions given in [KSS09]. The circuit for comparing ℓ bit integers requires 4ℓ non-XOR gates. The circuit for testing equality of ℓ bit integers also requires 4ℓ non-XOR gates. The circuit for adding two ℓ bit integers requires 4ℓ non-XOR gates, while the circuit for multiplying two ℓ bit integers requires $8\ell^2 - 4\ell$ non-XOR gates.

3 Proving non-Algebraic Statements on Algebraic Commitments

An important sub-protocol used in our constructions, is to commit to an input x using an algebraic commitment $\mathsf{Com}(x)$ (e.g. pedersen commitment), and perform a zero-knowledge proof of a non-algebraic statement about x, i.e. that $f(x) = 1$ for a boolean circuit f.

Such a protocol allows one to efficiently switch between proving algebraic statements on a committed input (e.g. proof of knowledge of a signature on a committed input) and non-algebraic statement (e.g. hashing, comparison, equality testing and more).

All our protocols are defined in terms of an ideal functionality, and are proven secure in the ideal/real world paradigm. We start by defining this task in terms of an ideal functionality in Figure 2. We provide two instantiations for this functionality that provide different efficiency trade-offs depending on the input size and the algebraic commitment scheme used.

Fig. 2. The ideal functionality $\mathcal{F}_{\mathsf{Com},f}$

- The verifier inputs $\mathsf{Com}(x)$ and prover inputs the opening information x and the randomness.
- If $f(x) = 1$ and the opening to the commitment verifies, output accept to the verifier.

The starting point for both instantiations is the ZK-proof of non-algebraic statements based on garbled circuits [JKO13] (see section 2.6). As the naive solution we

could garble a circuit that takes x and the opening of $\mathsf{Com}(x)$ as prover's input and outputs 1 if $f(x) = 1$ and $\mathsf{Com}(x)$ correctly opens to x. The main drawback of this solution is that checking correctness of opening for an algebraic commitment requires performing expensive group operations (e.g. exponentiation) inside the garbled circuit which would dominate the computation/communication cost. Our two instantiations show how to avoid these costs and perform all algebraic operations outside the garbled circuit.

3.1 First Instantiation

In our first construction, we have the prover commit to each bit of x, i.e. $\mathsf{Com}(x_i)$ for all $i \in [n]$, and prove that when combined they yield x.

Then, following the GC-based approach, the verifier constructs a garbled circuit that computes $f(x)$, parties go through the steps of the GC-based ZK proof for the prover to prove knowledge of a value x' such that $f(x') = 1$. The main issue is that a malicious prover may use a different input $x' \neq x$ in the circuit than what he committed to.

But we observe that the input keys associated with x' in the GC (which are obtained through the COT), can function as one-time MACs on each bit of x' and can be used to enforce that $x' = x$ using efficient algebraic ZK proofs that take place outside the garbled circuit. In particular, immediately after the COTs, the prover commits to its input keys i.e. $K_i^{x'_i}$ for the ith bit of x'. When the GC is opened and both input keys K_i^0, K_i^1 are opened, the prover can provide ZK proofs that $K_i^{x'_i} = x_i K_i^1 + (1 - x_i) K_i^0$ if the commitment scheme provides efficient proofs of linear relations.

The complete protocol description in the COT-hybrid model is given in Figure 3. We point out that steps 1, 6 and 13 are additions compared to the protocol of [JKO13].

Theorem 1. *Let $\mathcal{G}$ be a garbling scheme satisfying correctness and authenticity properties as defined in 2.4. Let* Com *be a secure commitment scheme, and let the proofs PK be implemented with a zero knowledge proof of knowledge. Then, the protocol $\Pi_{\mathsf{Com},f}$ in Figure 3 securely implements $\mathcal{F}_{\mathsf{Com},f}$ in the presence of malicious adversaries in the $\mathcal{F}_{COT}$-hybrid model.*

Proof. **Corrupt Prover.**

The simulator works as follows: It uses the OT simulator to extract the prover's input x' to the OT. It then plays the role of the honest verifier in the rest of the simulation - it constructs the garbled circuit honestly and uses its input keys as verifier's inputs to the COT functionality. The simulator then extracts the value Z' committed to by the prover from the proofs of knowledge of opening in step 8. It also extracts prover's committed input x and the values K'_i that prover committed to in the protocol, using the extractor for the ZK proof of knowledge in step 13. The simulator finally outputs x and the opening extracted from the ZK proofs, iff all the following hold: $x = x'$, $f(x) = 1, Z$ is the one-key of the output wire, $K'_i = K_i^{x_i}$ for all i, the commitment in step 8 is opened to Z, and the ZK proofs of step 13 verifies. Note that in the ideal model the functionality will always output accept when the simulator sends this witness.

We now prove that a corrupt prover's view in the real protocol is indistinguishable from his view with the simulator via a series of intermediate games.

Fig. 3. The Protocol $\Pi_{\mathsf{Com},f}$

Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Eval}, \mathsf{Ve})$ be a verifiable garbling scheme. Let F be the following functionality: it takes as input x, and outputs v such that $v = 1$ if $f(x) = 1$ and 0 otherwise. The prover has input x, the verifier is in possession of $C_x = \mathsf{Com}(x)$ and both parties have as input the security parameter κ.

1. The prover commits to the bits of x by sending bit-wise commitment to x: $C_i = \mathsf{Com}(x_i), \forall 1 \leq i \leq n$.
2. The verifier constructs a garbled circuit for F.

$$(GC, e, d) \leftarrow \mathsf{Gb}(1^\kappa, F)$$

3. The prover inputs his choice bits by sending (i, x_i) for all $i \in [n]$ to $\mathcal{F}_{COT}$.
4. The verifier inputs the wire labels corresponding to the prover's input by sending (i, K_i^0, K_i^1) for all $i \in [n]$ to $\mathcal{F}_{COT}$.
5. $\mathcal{F}_{COT}$ outputs K_i' for all $i \in [n]$ to the prover where $K_i' = K_i^{x_i}$.
6. The prover commits to the received input wire labels by sending $C_{K_i} = \mathsf{Com}(K_i')$ for all i.
7. The prover evaluates the garbled circuit

$$Z \leftarrow \mathsf{Eval}(GC, \{K_i'\}_{i \in [n]})$$

8. Prover commits to the garbled output Z by sending $\mathsf{Com}(Z)$ to the verifier and proves knowledge of opening.
9. Verifier sends open to $\mathcal{F}_{COT}$.
10. $\mathcal{F}_{COT}$ sends (K_i^0, K_i^1) to the prover for all $i \in [n]$.
11. Prover verifies that the correct circuit was garbled by running $\mathsf{Ve}(GC, \{K_i^0, K_i^1\}_{i \in [n]})$. If the output is not accept, the prover terminates. Otherwise if Ve outputs accept, he opens the commitment to the output Z by sending Z and the randomness used in $\mathsf{Com}(Z)$.
12. Verifier checks that the opening is correct and that $\mathsf{De}(d, Z) = 1$. If the opening is not correct or if $\mathsf{De}(d, Z) \neq 1$, the verifier outputs reject and terminates.
13. If the verifier did not terminate, the prover and the verifier engage in a Zero-knowledge protocol to prove the following:
    - $\mathsf{PK}\{(x_i, K_i', r, R) : C_i = \mathsf{Com}(x_i) \wedge C_{K_i} = \mathsf{Com}(K_i') \wedge K_i' = x_i K_i^1 + (1 - x_i) K_i^0\}, \forall 1 \leq i \leq n$.
    - $\mathsf{PK}\{(x, x_1, \cdots, x_n, r, r_1, \cdots r_n) : C_x = \mathsf{Com}(x) \wedge C_i = \mathsf{Com}(x_i) \wedge x = \sum 2^i x_i\}$
14. If the zero-knowledge proof verifies, the verifier outputs accept.

- Game Ideal: This is the interaction of the corrupt prover with the simulator and functionality as described above.
- Game G_0: This is the interaction of the corrupt prover with the simulator as described above, with the exception that instead of the simulator sending x and the opening to F, which outputs accept iff $f(x) = 1$, the game will output accept iff $f(x') = 1$ for the x' extracted from the OT (and all the other conditions listed hold). Since one of the conditions checks $x = x'$, this is identical.

- Game G_1: This game, behaves exactly as in G_0 except for a slight change in the accept condition. It outputs accept if $f(x') = 1$ and $K'_i = K_i^{x_i}$ for all i and Z is the one-key of the output wire and the commitment in step 8 is correctly opened to Z, and all the ZK proofs verify (i.e. no $x = x'$ check).
  Indistinguishability:
  Define the event Bad as the event that $x \neq x', f(x') = 1, Z$ is the one-key of the output wire, $K'_i = K_i^{x_i}$ for all i, and the opening to Z is correct and the ZK proofs of step 13 verify.
  Observe that G_0 is identical to G_1 conditioned on $\overline{\mathsf{Bad}}$. We now argue that $\Pr[\mathsf{Bad}]$ is negligible, by observing that an adversary who makes us reject G_0 but accept in G_1, can only succeed with probability $1/2^s$ where s is a statistical security parameter, given the COT hybrid model. Without loss of generality lets assume the ith bit of x is 0 and ith bit of x' is 1. Then, the probability of the adversary guessing K_i^0 given only K_i^1 is less than $1/2^{|K_i^0|}$. Note that $|K_i^0|$ is the computational security parameter, which is 128 bits for an AES key. But without loss of security we can used a truncated K_i^0 (to its least significant s bits) in the ZK proofs of step 13.
  Hence Games G_0 and G_1 are indistinguishable except with negligible probability in s.
- Game G_2: This game behaves as in G_1 except for another change in the accept condition. We accept if $f(x') = 1$ and ZK proofs of step 13 verifies and Z is the one-key of the output wire, and the commitment in step 8 is correctly opened to Z (i.e. no $K'_i = K_i^{x_i}$ check).
  If an adversary can distinguish between Games G_1 and G_2, we can break the *soundness of the ZK proof* of step 13. Therefore, G_1 and G_2 are indistinguishable.
- Game G_3: This game behaves as in G_2 except for a small change in accept condition. We accept if ZK proofs of step 13 verifies and Z is the one-key of the output wire, and the commitment in step 8 is correctly opened to Z (i.e. no $f(x') = 1$ check).
  Games G_2 and G_3 are identical, except when the following event occurs: $f(x') \neq 1$ and ZK proof of step 13 passes, and Z is the one-key of the output wire. When this event occurs, we accept in G_3 and rejects in G_2. We now argue that the probability of this event is negligible. For the sake of contradiction, assume the prover's input to OT is x' such that $f(x') \neq 1$, but the value committed to is the correct one-key Z for the output wire. We can use such a prover to break the authenticity of the garbling scheme (See definition 2).
- Game G_4: This game behaves as in G_3 except for the accept condition. We accept if the ZK proofs of step 13 verifies and the the commitment in step 8 opens correctly (i.e. no check that it is the same as extracted Z).
  An adversary who can distinguish between G_3 and G_4 can be used to violate the binding property of the commitment scheme.
  G_4 is identical to the real world game with an honest verifier.

Corrupt Verifier. The simulator commits to bits of a random value. It also uses a random value as prover's inputs to the COT, and receives the verifier's inputs to the COT functionality (K_i^0, K_i^1) for all i, i.e. the input keys to the verifier GC. The simulator then commits to the keys corresponding to the random input it used in the OTs.

It then runs $\mathsf{Ve}(GC, (K_i^0, K_i^1), f)$ to either obtain reject, or the decoding information d. If the output is reject it commits to a dummy value, else it commits to the one-key for the output wire, denoted by Z.

It then receives the "open" message from the verifier. If Ve had not output reject earlier, the simulator opens the commitment to Z and uses the simulator for the ZK proof to simulate the proofs of step 13. Otherwise, the simulator aborts.

- Game G_0: This is the interaction of the corrupt verifier with the simulator as described above.
- Game G_1: Is similar to game G_0 except that the real input x of prover is committed to.
  The two games are indistinguishable due to the hiding property of the commitment scheme.
- Game G_2: Is similar to G_1 except that instead of computing Z by running Ve, we run $\mathsf{Eval}(GC, K_i^{x_i})$ to compute and commit to Z.
  The two games are indistinguishable due to the second condition in the *correctness* property of the garbling scheme. Note that we are also implicitly using the committing OT property (the protocol described in the COT hybrid model) as the keys extracted in the OTs and what the functionality sends to the honest prover are the same.
- Game G_3: Is similar to G_2 except that the honest input x of the prover is used in the OTs.
  The two games are identical in the OT hybrid model.
- Game G_4: Is similar to G_3 except that the simulator commits to inputs keys associated with the real input x.
  The two games are identical due to the hiding property of the commitment scheme.
- Game G_5: Is similar to G_4 except that in step 13, the simulator performs the proofs, honestly.
  The two games are indistinguishable due to zero-knowledge property of the ZK proof.
  Note that G_5 is the real game with the honest prover.

3.2 Second Instantiation

We now give an alternative construction that implements the functionality in Figure 2. In particular, we avoid the bit-wise commitments to each bit of x_i, and the associated bit-wise ZK proofs, and hence require fewer public-key operations (exponentiations) in the construction. On the other hand, the garbled circuit is augmented and hence a larger number of symmetric-key operations are needed.

The idea is as follows. In order to ensure that the prover uses the same input x in the GC, we have the circuit not only compute $f(x)$ but also a one-time MAC of x, i.e. $t = ax + b$ for random a and b of the verifier's choice. a and b are initially unknown to the prover, but are opened along with the GC after the prover has committed to t. Given a and b, the prover then provides a ZK proof that $\mathsf{Com}(t)$ is indeed the one-time MAC of x (using efficient proofs of linear relations). We note that the $t = ax + b$ operation performed in the circuit is on integers.

We note that our construction deviates from the standard construction of GC-based ZK where the verifier has no input, and privacy-free garbling is sufficient. In particular, we do invoke the privacy property of the garbling scheme in our construction to ensure that the prover does not learn a and b, until the opening stage.

The complete protocol description in the COT-hybrid model is given in Figure 4.

Theorem 2. *Let $\mathcal{G}$ be a garbling scheme satisfying correctness, authenticity, and privacy properties as defined in section 2.4. Let* Com *be a secure commitment scheme, and let the proofs PK be implemented with a zero knowledge proof of knowledge. Then, the protocol $\Pi_{\mathsf{MAC},f}$ in figure 4 securely implements $\mathcal{F}_{\mathsf{Com},f}$ in the presence of malicious adversaries in the $\mathcal{F}_{COT}$-hybrid model.*

Proof. **Corrupt Prover.**

The simulator works as follows: It uses the OT simulator to extract the prover's input x' to the OT. It then plays the role of the honest verifier in the rest of the simulation - it chooses a, b randomly as the honest verifier would, constructs the garbled circuit honestly and uses its input keys as verifier's inputs to the COT functionality. The simulator then extracts the value Z' committed to by the prover from the proofs of knowledge of opening in step 8. It also extracts prover's committed input x and the tag t' that the prover committed to in the protocol, using the extractor for the ZK proof of knowledge in step 16. The simulator finally outputs x and the opening extracted from the ZK proofs, iff all the following hold: $x = x'$, $f(x) = 1, Z$ is the one-key of the output wire, $t' = ax + b$, the commitment in step 8 is opened to Z, and the ZK proof of step 16 verifies. Note that in the ideal model the functionality will always output accept when the simulator sends this witness.

We now prove that a corrupt prover's view in the real protocol is indistinguishable from his view with the simulator by a series of intermediate games.

- Game Ideal: This is the interaction of the corrupt prover with the simulator and functionality as described above.
- Game G_0: This is the interaction of the corrupt prover with the simulator as described above, with the exception that instead of the simulator sending x and the opening to F, which outputs accept iff f(x)=1, the game will output accept iff $f(x') = 1$ for the x' extracted from the OT (and all the other conditions listed hold). Since one of the conditions checks $x = x'$, this is identical.
- Game G_1: In this game, the simulator behaves exactly as in G_0 except that it does not check the $x = x'$ condition.
  Define the event Bad as the event that $x \neq x'$ but $t = ax + b$. Observe that G_0 is identical to G_1 conditioned on $\overline{\mathsf{Bad}}$. We argue that $\Pr[\mathsf{Bad}]$ is negligible due to the unforgeability property of the one-time MAC, the hiding property of the commitment scheme, and the privacy of the garbled circuit.
  Consider a game where we run as in G_1 but stop after step 10, and look at the probability that in this gane $t' = ax + b$ but $x \neq x'$; if $\Pr[\mathsf{Bad}]$ is nonnegligible, this will be nonnegligible as well. Now, by the privacy of the garbled circuit, this is indistinguishable from a game where the verifier computes a tag t on x', and then constructs (GC, e, d) using the privacy simulator: $\mathcal{S}(F, (t, 1))$. Similarly, by the hiding of the commitment scheme this is still indistinguishable from a game where

the verifier commits to random values instead of a, b. But if in this final game we get $t' = ax + b$ and $x \neq x'$ with non-negligible probability, then we can break the unforgeability of the MAC. The probability of forgery is bounded by $1/2^{|a|}$, and hence exponentially small in the statistical security parameter $s = |a|$.

- Game G_2: In this game, the simulator behaves as in G_1 except that it does not check the condition $t = ax + b$.
  If an adversary can distinguish between Games G_2 and G_1, we can break the soundness of the ZK proof of step 16.
- Game G_3: In this game, the simulator behaves as in G_2 except that we do not check the condition $f(x') = 1$.
  Games G_2 and G_3 are identical, except when the following event occurs: $f(x') \neq 1$ and ZK proof of tag verifies and Z is the one-key of the output wire. We now argue that the probability of this event is negligible. For the sake of contradiction, assume the prover's input to OT is x' such that $f(x') \neq 1$, but the value committed to is the correct one-key Z for the output wire. We can use such a prover to break the authenticity of the garbling scheme (See definition 2).
- Game G_4: In this game, the simulator behaves as in G_3 except for the accept condition. The simulator accepts if the ZK proofs of step 16 verifies and the commitment in step 8 opens correctly (i.e. no check that it is the same as extracted Z).
  An adversary who can distinguish between G_4 and G_3 can be used to violate the binding property of the commitment scheme.
  G_4 is identical to the real world game with an honest verifier.

Corrupt Verifier. The simulator extracts a and b from the proofs of knowledge of their openings by verifier. It uses a random value as prover's inputs to the COT, and receives the verifier's inputs to the COT functionality (K_i^0, K_i^1) for all i, i.e. the input keys to the verifier GC.

It then runs $\mathsf{Ve}(GC, (K_i^0, K_i^1), F)$ (and checks against the extracted a, b) to either obtain reject, or the decoding information d. If the output is reject it commits to dummy values for Z and t, else it commits to the one-key for the output wire denoted by Z, and dummy t.

The simulator receives the openings of $\mathsf{Com}(a)$ and $\mathsf{Com}(b)$. If the openings are not what it extracted earlier, or if Ve had output reject earlier, it aborts. Else, the simulator opens the commitment to Z and uses the simulator for the ZK proof to simulate the proofs of step 16.

- Game G_0: This is the interaction of the corrupt verifier with the simulator as described above.
- Game G_1: Is similar to game G_0 except that $t = ax + b$ for the real input x of prover is committed to.
  The two games are indistinguishable due to the hiding property of the commitment scheme.
- Game G_2: Is similar to G_1 except that instead of computing Z and t by running Ve, we run $\mathsf{Eval}(GC, K_i^{x_i})$ to compute and commit to Z and t.
  The two games are indistinguishable due to the second condition in the *correctness* property of the garbling scheme, and binding property of commitments $\mathsf{Com}(a)$

and $\mathsf{Com}(b)$. Note that we are also implicitly using the committing OT property (the protocol described in the COT hybrid model) as the keys extracted in the OTs and what the functionality sends to the honest prover are the same.

- Game G_3: Is similar to G_2 except that the honest input x of the prover is used in the OTs.
  The two games are identical in the OT hybrid model.
- Game G_4: Is similar to G_3 except that in step 13, the simulator performs the proofs honestly.
  The two games are indistinguishable due to zero-knowledge property of the ZK proof.
  Note that G_4 is the real game with the honest prover.

3.3 Efficiency Comparison and Optimizations

Efficiency Comparison In our first instantiation, in addition to the cost associated with the GC-based ZK, i.e. the oblivious transfer for x and the cost of garbling f, $O(n)$ exponentiations are necessary to commit to each bit of input x and to perform the bitwise ZK proofs associated with them in the last step.

In our second instantiation, the bitwise commitments/proofs are eliminated (i.e. only a constant number of exponentiations) but instead the circuit for $ax + b$ needs to be garbled which requires $O(ns + s^2)$ additional symmetric-key operations when using textbook multiplication (we discuss range of values for s shortly). Using Karatsuba's multiplication algorithm [Knu69], this can potentially be further reduced.

The round complexity of both protocols is essentially the same as the GC-based ZK proof of [JKO13] (5 rounds), as the extra messages can be sent within the same rounds. (To simplify presentation, we used a separate step for each operation in our protocol description, but many of these can be combined.) A more round-efficient GC-based ZK proof would make our constructions more round efficient as well.

The first instantiation requires more exponentiations which are significantly costlier than their symmetric-key counterpart, but the total number of symmetric-key operations in the second instantiation is higher. Hence, when n is small, the first instantiation is likely more efficient, while when n is larger, the second instantiation will be the better option. Furthermore, if bit-wise commitment to the input is already necessary as part of the bigger protocol (as is the case in some of our constructions), the first instantiation may be the better choice. In the case where a comparison circuit $x < q$ is also computed, an additional $O(n)$ symmetric-key operations suffices.

Optimizations Next we review a few other optimizations that improve efficiency of our instantiations.

- *Reducing exponentiations.* We consider the following optimization for the protocol $\Pi_{\mathsf{Com},f}$ in Fig. 3 which reduces the number of exponentiations necessary for the ZK proofs significantly. In step 6, the prover commits to the sum of the keys received instead of individually to each wire key. The prover sends $\mathsf{Com}(S) = \mathsf{Com}\left(\sum_{i=1}^{n} K'_i\right)$ in step 6. We assume that the bit commitment scheme Com is homomorphic, and each wire key K_i is truncated to s bits and interpreted as a group

element. Now, in the zero knowledge proofs of step 13, the prover proves the following statements which can be performed with fewer exponentiations:

- $\mathsf{PK}\{(x_i, S, r, R) : \mathsf{Com}(x_i) = g^{x_i}h^r \wedge \mathsf{Com}(S) = g^S h^R \wedge S = \sum_{i=1}^{n} \left(x_i K_i^1 + (1 - x_i)K_i^0\right)\}$
- $\mathsf{PK}\{(x, x_1, \cdots, x_n, r, r_1, \cdots r_n) : \mathsf{Com}(x) = g^x h^r \wedge \mathsf{Com}(x_i) = g^{x_i}h^{r_i} \wedge x = g^{\sum 2^i x_i} h^r\}$

We can show that if the sum extracted by the simulator from the commitment in step 6 is not equal to the sum of keys corresponding to the input x' extracted from COT, but the ZK proofs verify, then for some i, the prover must have correctly guessed K_i^b such that $b \neq x'_i$. The probability of this is negligible by the security of the COT protocol.

– *Privacy-free garbling.* As discussed earlier, in [FNO15] it is observed that privacy-free garbling is sufficient for GC-based ZK proofs of non-algebraic statements. This improves the communication/computation cost of garbled circuits in our first instantiation by a factor of two. But as mentioned earlier, the same cannot be said about our second construction since the privacy property of garbling is required to hide a and b in the earlier stage of the construction.
But we can think of bigger circuit as consisting of two smaller circuits: one computing the function f and the other computing $ax + b$. If we split the computation into two garbled circuits with shared OT, then we can use the privacy free garbling scheme of [FNO15,ZRE15] for the first circuit as the verifier has no input, and use a standard garbling scheme for the $ax + b$ circuit.
– *Smaller multiplication circuit.* For the one-time MAC in the second protocol, a small a is sufficient for security - if the security (unforgeability) desired is 2^{-s}, it suffices for a to be s bits long. Hence, for a 512-bit input, a $40-80$-bit a is sufficient to compute $ax + b$ which reduces the size of the multiplication circuit significantly.

3.4 Secure computation on committed/signed inputs

In the protocols described above, we have shown how to commit to a value $\mathsf{Com}(x)$ and then use a GC-based ZK proof to prove non-algebraic statements about x.

It is not hard to show that one can extend this approach, to a full-fledged secure two-party computation (2PC) of any function $g(x, y)$ where x is the committed input of the prover. In particular, note that in the ZK proof, the prover feeds its input x into the COTs in order to obtain its inputs keys to the GC of the ZK proof. In order to extend this to a secure 2PC based on garbled circuits, we let the prover play the role of the evaluator in a cut-and-choose 2PC based on garbled circuits, and use the same COT as above for the prover to obtain the garbled inputs for x in the 2PC. This would ensure that the same x that was used in the ZK proof is also used in the 2PC, and the ZK proof already ensures that this is the same input committed to in $\mathsf{Com}(x)$.

A subtle point here is that we need to open the sender's input to the COTs for the GC for the ZK but not for the GCs for the 2PC. This is supported by the committing OT of [S$^+$11] (also see the discussion on COTs in [MR13]). It is interesting to explore the use of OT extension in such COTs where some sender inputs are opened while others are not.

We emphasize that the GCs for the 2PC only garble the desired function g, and hence the GC for the ZK proof is not part of any cut-and-choose. However, we note that the above technique is currently limited to the evaluator's input since the OTs for evaluator's input enable an almost-free check of equality of inputs in the 2PC and the ZK. Extending the ideas to both party's inputs is an interesting future direction.

This approach can be easily extended to prove other statements about x, such as proof of knowledge of a signature on x (hence signed-input 2PC) either using the techniques we give below in the case of RSA/DSA signatures, or using previous techniques to give a proof of knowledge of a CL signature[CL01].

4 Building Blocks for Privacy-Preserving Signature Verification

We introduce three important building blocks for our privacy-preserving signature verification protocols. Two of them can be directly instantiated using our $\mathcal{F}_{\mathsf{Com},f}$ functionality introduced in section 3, while for the third one we provide a customized construction.

4.1 Proving that a committed value is the hash of another committed value

Here, the goal is to commit to a message m and its hash $\mathcal{H}(m)$ and prove in zero-knowledge that one committed value is the hash of the other. We define the task in terms of an ideal functionality in Figure 5.

Fig. 5. The ideal functionality $\mathcal{F}_{Hash}$

- The verifier inputs $\mathsf{Com}(m), \mathsf{Com}(M)$ and the prover inputs the opening information (m, M) and the randomness.
- If $\mathcal{H}(m) = M$ and the openings to the commitments verify, output accept to the verifier.

We now use the abstract functionality $\mathcal{F}_{\mathsf{Com},f}$ from Fig 2 with a commitment scheme Com_h to instantiate a protocol that implements $\mathcal{F}_{Hash}$. Here, the input is $x = (m, M = \mathcal{H}(m))$ and the Com_h is defined as $\mathsf{Com}_h(x = (m, M)) = (\mathsf{Com}(m), \mathsf{Com}(M))$. To commit to bits of x, one can commit to bits of m and M individually. Com_h inherits efficient proofs of linear relations from Com as long as the proofs on m and M are performed separately. Given these, we show in Figure 6 how to implement $\mathcal{F}_{Hash}$ by defining the right function f for the ideal functionality $\mathcal{F}_{\mathsf{Com},f}$.

Fig. 6. The Protocol Π_{Hash}

1. The prover commits to $x = (m, M)$ by sending $\mathsf{Com}_h(x) = \mathsf{Com}(m), \mathsf{Com}(M)$ to the verifier.
2. The prover and the verifier run $\Pi_{\mathsf{Com},f}$ where f is the following functionality: f takes m and M as inputs and outputs v such that $v = 1$ if $\mathcal{H}(m) = M$ and 0 otherwise.

Theorem 3. *The protocol Π_{Hash} in figure 6 securely implements $\mathcal{F}_{Hash}$, given the ideal functionality $\mathcal{F}_{\mathsf{Com},f}$, in the presence of malicious adversaries.*

4.2 Proof of equality of committed values in different groups

The goal is to prove that the value committed to in different prime groups of size p and q are the same. We define the task in terms of an ideal functionality, defined in Figure 7. This can be achieved using standard techniques which involve using the integer commitment scheme by Damgard and Fujisaki [DF02] to prove properties about the discrete logarithms in $\mathbb{Z}$ (instead of modulo the order of the group). This requires that the verifier choose an RSA modulus $\widetilde{N}$ such that the factorization is unknown to the prover, and prove that it is chosen correctly in an initial set-up phase. The prover also has to compute exponentiations in an RSA group where the exponents are $|\widetilde{N}| + \kappa$ bits long. Since the group order is hidden, chinese remaindering cannot be used to speed up the exponentiations, and therefore the approach is fairly expensive. We give a protocol that avoids the integer commitment technique.

Fig. 7. The ideal functionality $\mathcal{F}_{Eq}$

- The verifier inputs $\mathsf{Com}_p(x), \mathsf{Com}_q(y)$ and the prover inputs (x, y) and the opening information. p and q are public primes and $q < p$.
- If $0 \leq x < p, 0 \leq y < p, x \equiv y \mod q$, and the openings to the commitments verify, output accept to the verifier.

In Figure 8, we use the ideal functionality $\mathcal{F}_{\mathsf{Com},f}$ from Fig 2 with a commitment scheme Com_{pq} to instantiate a protocol that implements $\mathcal{F}_{Eq}$. The scheme is defined as $\mathsf{Com}_{pq}(x) = (\mathsf{Com}_p(x), \mathsf{Com}_q(x))$, where it is assumed that Com_p and Com_q allow for proving linear relationships among committed values.

Fig. 8. The Protocol Π_{Eq}

1. The prover commits to x and y by sending $\mathsf{Com}_p(x), \mathsf{Com}_q(y)$ to the verifier.
2. The prover and the verifier run $\Pi_{\mathsf{Com},f}$ where f is the following functionality: f takes x and checks that it is upper bounded by p and outputs v such that $v = 1$ if $x \leq p$ and 0 otherwise.

4.3 Proof of equality of discrete logarithm of a committed value and another committed value

Let $\mathbb{G}_1 = \langle G_1 \rangle$ and $\mathbb{G}_2 = \langle G_2 \rangle$ be two groups of order p and q respectively with $q|p-1$ and let $g \in \mathbb{G}_2$ be an element of order q . Given $y_1 = G_1^{g^x} H_1^{R_1}$ and $y_2 = G_2^x H_2^{R_2}$, we want to prove that the discrete logarithm w.r.t to base g of the value committed to in y_1 is equal to the value committed to in y_2. Let k be a security parameter. Following

standard notation, we denote the protocol by $\mathsf{PK}\{(x, R_1, R_2) : y_1 = G_1^{g^x} H_1^{R_1} \wedge y_2 = G_2^x H_2^{R_2}\}$. The technique of our protocol is similar to [Sta96], [CS97a], and is a variant of [MGGR13]. Our protocol is only honest verifier zero-knowledge. This HVZK protocol can be compiled into a full zero-knowledge proof of knowledge in the auxiliary string model using the technique of [Dam00].

Fig. 9. $\mathsf{PK}\{(x, R_1, R_2) : y_1 = G_1^{g^x} H_1^{R_1} \wedge y_2 = G_2^x H_2^{R_2}\}$

Given $y_1 = G_1^{g^x} H_1^{R_1}$ and $y_2 = G_2^x H_2^{R_2}$

1. The prover computes the following $2k$ values: $u_i = G_1^{g^{\alpha_i}} H_1^{\beta_i}$ and $v_i = G_2^{\alpha_i} H_2^{\gamma_i}$ for $1 \leq i \leq k$, for randomly chosen $\alpha_i, \gamma_i \in \mathbb{Z}_q$ and $\beta_i \in \mathbb{Z}_p$, and sends u_i, v_i to the verifier.
2. The verifier chooses a random string c of length k as the challenge, and sends it to the prover.
3. For a challenge string $c = c_1 \ldots c_k$, compute and send the tuple (r_i, s_i, t_i)
   If $c_i = 0$,
   $$r_i = \alpha_i, s_i = \beta_i, t_i = \gamma_i$$
   If $c_i = 1$,
   $$r_i = \alpha_i - x \pmod q, s_i = \beta_i - R_1 g^{r_i} \pmod p, t_i = \gamma_i - R_2 \pmod q$$
4. Verification:
   If $c_i = 0$, check whether $u_i = G_1^{g^{r_i}} H_1^{s_i}$ and $v_i = G_2^{r_i} H_2^{t_i}$
   If $c_i = 1$, check if $u_i = y_1^{g^{r_i}} H_1^{s_i}$ and $v_i = y_2 G_2^{r_i} H_2^{t_i}$. The verifier accepts if Verification succeeds for all i.

We will show that the protocol in Figure 9 is correct, has a soundness error of $1/2^k$, and is honest verifier zero knowledge.

Proof. – **Completeness:** If the prover and the verifier behave honestly, it is easy to see that verification conditions hold.
If $c_i = 0$:
$$G_1^{g_i^r} H_1^{s_i} = G_1^{g^{\alpha_i}} H_1^{\beta_i} = u_i \text{ and } G_2^{r_i} H_2^{t_i} = G_2^{\alpha_i} H_2^{\gamma_i} = v_i$$
If $c_i = 1$:
$$y_1^{g^{r_i}} H_1^{s_i} = (G_1^{g^x})^{g^{r_i}} (H_1^{R_1})^{g^{r_i}} H_1^{s_i} = G_1^{g^{\alpha_i}} H_1^{\beta_i} = u_i \text{ and}$$
$$y_2 G_2^{r_i} H_2^{t_i} = G_2^x H_2^{R_2} G_2^{r_i} H_2^{t_i} = v_i$$

– **Soundness:** We show an extractor that computes x, R_1, R_2 given two different accepting views with same commitments but different challenge strings. Say, we have two accepting views for challenges c and $\hat{c} \neq c$. Without loss of generality, let us assume that they differ in the jth position, and $c_j = 0$. We have,
$$u_j = G_1^{g^{r_j}} H_1^{s_j} = y_1^{g^{\hat{r}_j}} H_1^{\hat{s}_j}$$

$$G_1^{g^{r_j}} H_1^{s_j} = G_1^{g^x g^{\hat{r_j}}} H_1^{Rg^{\hat{r_j}} + \hat{s_j}}$$

$$g^x = g^{r_j - \hat{r_j}}$$

We can compute (in $\mathbb{Z}_q$),

$$x = r_j - \hat{r_j}$$

We have,

$$s_j = R_1 g^{\hat{r_j}} + \hat{s_j}$$

and thus,

$$R_1 = \frac{s_j - \hat{s_j}}{g^{\hat{r_j}}}$$

We also have

$$v_j = G_2^{r_j} H_2^{t_j} = y_2 G_2^{\hat{r_j}} H_2^{\hat{t_j}}$$

$$G_2^{r_j} H_2^{t_j} = G_2^{x + \hat{r_j}} H_2^{\hat{t_j} + R_2}$$

and thus,

$$R_2 = t_j - \hat{t_j}$$

– **Honest Verifier Zero Knowledge:** We show a simulator such that the output of the simulator is statistically indistinguishable from the transcript of the protocol with a prover. The simulator on input c, randomly chooses $\alpha_i = r_i \in \mathbb{Z}_q, \beta_i = s_i \in \mathbb{Z}_p, \gamma_i = t_i \in \mathbb{Z}_q$ and computes for $1 \leq i \leq k$:
If $c_i = 0$,

$$u_i = G_1^{g^{r_i}} H_1^{s_i} \text{ and } v_i = G_2^{r_i} H_2^{t_i}$$

if $c_i = 1$,

$$u_i = y_1^{g^{r_i}} H_1^{s_i} \text{ and } v_i = y_2 G_2^{r_i} H_2^{t_i}$$

5 Privacy-Preserving FDH-RSA Signature Verification

The FDH-RSA Scheme. The Full Domain Hash RSA signature scheme $\mathsf{FDH} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is defined as follows [BR93]. The KeyGen algorithm on input the security parameter k, selects two $k/2$-bit primes p and q and computes the modulus $N = pq$. It then chooses an exponent $e \in \mathbb{Z}^*_{\phi(N)}$, and computes d such that $ed = 1 \mod \phi(N)$. Return (pk, sk), where $pk = (N, e)$ and $sk = (N, d)$. The signature generation and verification are as follows and use a hash function $\mathcal{H} : \{0,1\} \to \mathbb{Z}^*_N$.

$\mathsf{Sign}_{N,d}(M)$
$x = \mathcal{H}(M)$
$\sigma = x^d \mod N$
return σ

$\mathsf{Verify}_{N,e}(M, \sigma)$
$y = \sigma^e \mod N$
$y' = \mathcal{H}(M)$
if $(y = y')$ **then return** 1;
else return 0;

5.1 Proof of Knowledge of RSA Signatures

Given $\mathsf{Com}_N(m)$, a commitment to m in a group of order N, the following protocol is a zero knowledge proof of knowledge of a valid RSA signature on m.

1. The prover has input (m, σ) and the verifier is in possession of $\mathsf{Com}_N(m) = C_1 = g^m h^{r_1}$
2. The prover commits to $M = \mathcal{H}(m)$, that is, $M \in \mathbb{Z}_N$, compute $\mathsf{Com}_N(M) = C_2 = g^M h^{r_2}$, for randomly chosen $r_2 \in Z_N^*$. Send C_2 to the verifier and prove knowledge of opening.
3. The prover and verifier engage in the protocol Π_{Hash} with inputs (m, M) and (C_1, C_2) respectively.
4. The prover proves knowledge of e-th root of a committed value([CS97a]). Given $y = C_2 = g^M h^r$, prover proves knowledge of σ, such that, $y = g^{\sigma^e} h^r$.

   (a) The prover computes the following tuple:

   $$(y_1, \cdots, y_{e-1}) \text{ where } y_i = g^{\sigma^i} h^{r_i}$$

   for randomly chosen $r_i \in \mathbb{Z}_N$, for $i = 1$ to $e - 1$.

   (b) The prover and the verifier run the following proof of knowledge:

   $$\mathsf{PK}\{(\alpha, (\beta_1, \cdots, \beta_e)) : y_1 = g^\alpha h^{\beta_1} \wedge y_2 = y_1^\alpha h^{\beta_2} \wedge \cdots \wedge y = y_{e-1}^\alpha h^{\beta_e}\}$$

When e is one greater than a power of 2, we can employ optimizations like repeated squaring to prove knowledge of e-th root. Given $y = g^{\sigma^e} h^r$, for $e = 2^k + 1$, step 4 in the verification protocol can be now be realized as follows:

1. The prover computes the following tuple:

   $$(y_0, y_1, \cdots, y_k) \text{ where } y_i = g^{\sigma^{2^i}} h^{r_i}$$

   for randomly chosen $r_i \in \mathbb{Z}_N$, for $i = 1$ to k.
2. The prover and the verifier run the following proof of knowledge:

$$\begin{aligned}\mathsf{PK}\{(\alpha, \alpha_1, \cdots, \alpha_k, \beta, \beta_0, \cdots, \beta_k, R_0, \cdots, R_k) :& \\ y_0 = g^\alpha h^\beta \wedge y_1 = y_0^\alpha h^{\beta_0} \wedge y_1 = g^{\alpha_1} h^{R_0} \wedge y_2 = y_1^{\alpha_1} h^{\beta_1}& \\ \wedge y_2 = g^{\alpha_2} h^{R_1} \cdots \wedge y_k = y_{k-1}^{\alpha_{k-1}} h^{\beta_{k-1}} \wedge y_k = g^{\alpha_k} h^{R_{k-1}} \wedge y = y_k^\alpha h^{\beta_k}\}&\end{aligned}$$

It might be possible to improve the efficiency for some e's by using addition chains for the integer e. An addition chain for integer e is an ascending sequence $1 = e_0 < e_1 < \cdots e_r = e$ such that for $1 \leq i \leq r$, we have $e_i = e_j + e_k$. The prover, now, would have to provide only the y_i's for which i is an element of the addition chain for e. The relations among the y_i's will be sightly different, but can be proved in a similar way.

The above verification protocol can also be adapted to support variants of RSA-based signatures, like the probabilistic signature scheme (PSS) from [BR96]. PSS is a probabilistic generalization of FDH which uses two hash functions and more complicated padding. We can instantiate protocol $\Pi_{\mathsf{Com},f}$ with an f that verifies the additional checks of PSS to achieve privacy preserving verification of a PSS signature.

5.2 Proof of security

We sketch a proof that the above protocol is a zero-knowledge proof of knowledge of an RSA signature on a committed message. The completeness follows easily from the security of protocol Π_{Hash}, and from the observation that

$$y = \left(y_{e-1}^{\alpha}\right) h^{\beta_e} = \left(\left(\cdots \left(g^{\alpha} h^{\beta_1}\right)^{\alpha} h^{\beta_2} \cdots\right)^{\alpha} h^{\beta_{e-1}}\right)^{\alpha} h^{\beta_e}$$

$$= g^{\alpha^e} h^{\beta_e + \alpha\beta_{e-1} + \cdots + \alpha^{e-1}\beta_1}$$

in step 4.

- Soundness: We show an extractor, that, given access to the prover, extracts (m, σ) such that $\mathsf{Verify}_{N,e}(m, \sigma) = 1$. The extractor invokes the simulator for the corrupt prover of protocol Π_{Hash} to extract m and M. It then runs the extractor corresponding to the proof in step 4b to extract α. By the security of Π_{Hash} and the binding property of Com, it follows that $\alpha^e \mod N = M = \mathcal{H}(m)$.
- Zero-knowledge: We sketch a simulator that simulates the verifier's view in the protocol. The simulator commits to a random value on behalf of the prover in step 2 by computing $C_2' = \mathsf{Com}(M')$. It sends C_2' to the verifier, proves knowledge of opening and invokes the simulator for the corrupt verifier of protocol Π_{Hash}. It then chooses $y_1, \cdots, y_{e-1} \in Z_N$ at random, and runs the simulator corresponding to the proof in step 4b. We can show that the view of the verifier in the protocol is indistinguishable from the view with the simulator.

6 Privacy-Preserving (EC)DSA Signature Verification

The DSA Scheme. The Digital Signature Algorithm (DSA) is a variant of the Elgamal signature scheme. The key generation, signature generation and verification algorithms are given next. The KeyGen algorithm chooses two primes p and q such that $q \mid p-1$. Let g be an element of order q in $\mathbb{Z}_p^*$. It then chooses x randomly from $\{1, \cdots, q-1\}$. The private key is set to be x and the public key is $(g, p, q, y), y = g^x \mod p$.

$\mathsf{Sign}(m)$	$\mathsf{Verify}(m, (r, s))$
$M \leftarrow \mathcal{H}(m)$	$M \leftarrow \mathcal{H}(m)$
Pick a random $k, 1 \leq k < q$	$w = s^{-1} \mod q$
$r = (g^k \mod p) \mod q$	$u_1 = Mw \mod q$
$s = k^{-1}(M + rx) \mod q$	$u_2 = rw \mod q$
return (r, s)	**if** $r = (g^{u_1} y^{u_2} \mod p) \mod q$ **then return**;
	1 **else return** 0;

The ECDSA Scheme. ECDSA is the elliptic curve analogue of DSA. It works in an elliptic curve group $E(\mathbb{Z}_p)$. The ECDSA Key generation, signature and verification algorithms are given below. The KeyGen algorithm chooses an elliptic curve E defined over $\mathbb{Z}_p$ such that the number of points in $E(\mathbb{Z}_p)$ is divisible by a large prime n. Pick

a point $P \in E(\mathbb{Z}_p)$ of order n. Let $d \in [1, n-1]$ be a randomly chosen integer. Set $Q = dP$. The public key is (E, P, Q, n) and the private key is d.

$\mathsf{Sign}(m)$
$M \leftarrow \mathcal{H}(m)$
Pick a random $k \in [1, n-1]$
$kP = (x_0, y_0)$
$r = x_0 \mod n$
$s = k^{-1}(M + rd) \mod n$
return (r, s)

$\mathsf{Verify}(m, (r, s))$
$M \leftarrow \mathcal{H}(m)$
if $r, s \notin [1, n-1]$ **then return**;
0
$w = s^{-1} \mod n$
$u_1 = Mw \mod n$
$u_2 = rw \mod n$
$(x_1, y_1) = u_1 P + u_2 Q$
$v = x_1 \mod n$
if $r = v$ **then return** 1;
else return 0;

6.1 Proof of Knowledge of DSA Signatures

Let (r, s) be the DSA signature on m. Let $\mathbb{G}_1 = \langle G_1 \rangle$ and $\mathbb{G}_2 = \langle G_2 \rangle$ be two distinct groups of order p and q respectively where p and q are the parameters of the DSA signature algorithm. One technical difficulty is that we have to show r in G_1 and G_2 is equal modulo q. For that purpose, we use our protocol Π_{Eq} from Figure 8 to prove equality across groups. We also employ our protocol from Fig. 9 to prove equality of discrete logarithm of a committed value and another committed value. We now describe the DSA verification protocol in detail. Given a commitment to m, the following protocol is a zero-knowledge proof of knowledge of a valid DSA signature on m.

1. The verifier is in possession of $C_1 = \mathsf{Com}_q(m)$, and the prover has as input message $(m, (r, s))$ and the opening information of C_1 to m.
2. The prover commits to $M = \mathcal{H}(m)$, that is, $M \in \mathbb{Z}_q$, compute $C_2 = \mathsf{Com}_q(M)$ Send C_2 to the verifier and prove knowledge of opening.
3. Now the prover and verifier engage in the protocol Π_{Hash} to prove that $M = \mathcal{H}(m)$.
4. The prover commits to the signature (r, s) by sending $\mathsf{Com}_{pq}(r) = (\mathsf{Com}_p(r), \mathsf{Com}_q(r))$ and $\mathsf{Com}_q(s)$. The prover also commits to the following values: $u_1 = \mathcal{H}(m)s^{-1}, u_2 = rs^{-1}, \alpha = g^{u_1}, \beta = y^{u_2}$, where g is the generator of a cyclic group of order q in $\mathbb{Z}_p^*$ used in DSA signing, and y is the DSA public key. Prover sends $\mathsf{Com}_q(u_1)$, $\mathsf{Com}_q(u_2), \mathsf{Com}_p(\alpha), \mathsf{Com}_p(\beta)$.
5. The prover and the verifier carry out the following Σ-protocol zero-knowledge proofs of knowledge:
   (a) $\mathsf{PK}\{(u_1, R_1, R_2) : \mathsf{Com}_p(\alpha) = G_1^{g^{u_1}} H_1^{R_1} \wedge \mathsf{Com}_q(u_1) = G_2^{u_1} H_1^{R_2}\}$
   (b) $\mathsf{PK}\{(u_2, R_1, R_2) : \mathsf{Com}_p(\beta) = G_1^{y^{u_2}} H_1^{R_1} \wedge \mathsf{Com}_q(u_2) = G_2^{u_2} H_1^{R_2}\}$
   (c) $\mathsf{PK}\{(r, \alpha, \beta, R_1, R_2, R_3) : \mathsf{Com}_p(\beta) = G_1^{\beta} H_1^{R_1} \wedge \mathsf{Com}_p(\alpha) = G_1^{\alpha} H_1^{R_2} \wedge \mathsf{Com}_p(r) = G_1^{r} H_1^{R_3} \wedge r = \alpha\beta\}$
   (d) $\mathsf{PK}\{(M, u_1, s, R_1, R_2, R_3) : \mathsf{Com}_q(M) = G_2^{M} H_2^{R_1} \wedge \mathsf{Com}_q(u_1) = G_2^{u_1} H_2^{R_2} \wedge \mathsf{Com}_q(s) = G_2^{s} H_2^{R_3} \wedge M = u_1 s\}$
   (e) $\mathsf{PK}\{(r, u_2, s, R_1, R_2, R_3) : \mathsf{Com}_q(r) = G_2^{r} H_2^{R_1} \wedge \mathsf{Com}_q(u_2) = G_2^{u_2} H_2^{R_2} \wedge \mathsf{Com}_q(s) = G_2^{s} H_2^{R_3} \wedge r = u_2 s\}$
6. The prover and verifier engage in Π_{Eq} with input $\mathsf{Com}_{pq}(r)$.

6.2 Proof of security

We sketch a proof of the soundness and zero-knowledge properties of the above protocol. The completeness follows from security of Π_{Hash} and completeness of the proofs of knowledge in step 5.

- Proof of Knowledge: We show an extractor, that, given access to the prover, extracts $(m, (r, s))$ such that $\mathsf{Verify}(m, (r, s)) = 1$. The extractor invokes the simulator for the corrupt prover of protocol Π_{Hash} to extract m and M and the opening information for C_1.
  It then runs the extractor guaranteed by the proof of knowledge property of the proofs in step 5 to extract $u_1, u_2, \alpha, \beta, s, r$. Finally it returns $(m, (r, s))$ and the opening information. By security of Π_{Hash}, Π_{Eq} and the binding property of the commitment scheme Com, it follows that $r = g^{Ms^{-1}} y^{rs^{-1}}$ and $M = \mathcal{H}(m)$.
- Zero-knowledge: We sketch a simulator that simulates the verifier's view in the protocol. The simulator commits to a random value on behalf of the prover in step 2 by computing $C_2' = \mathsf{Com}(M')$. It sends C_2' to the verifier, proves knowledge of the opening and invokes the simulator for the corrupt verifier of protocol Π_{Hash}. It then commits to random values in step 4, and runs the simulator corresponding to the proofs of knowledge in step 5. Finally in step 6, the simulator invokes the simulator for protocol Π_{Eq}. We can show that the view of the verifier in the protocol is indistinguishable from the view with the simulator.

6.3 Proof of Knowledge of ECDSA Signatures

Let (r, s) be the ECDSA signature on m. Let $\mathbb{G}_1 = \langle G_1 \rangle$ and $\mathbb{G}_2 = \langle G_2 \rangle$ be two distinct groups of order p and n respectively where p is the order of the field of the curve and n is the order of point P. Addition of elliptic curve points which is the group operation requires arithmetic operations in the underlying finite field $\mathbb{Z}_p$ of the curve E. We use a straight forward variant of the protocol in Fig. 9 to prove statements about multiples of an elliptic curve point (elliptic curve analogue of exponentiation) inside commitments.

1. The verifier is in possession of $C_1 = \mathsf{Com}_p(m)$ and the prover has as input (m, σ) and the opening of C_1 to m.
2. The prover commits to $M = \mathcal{H}(m)$, by computing $C_2 = \mathsf{Com}_p(M)$. Send C_2 to the verifier and prove knowledge of opening.
3. The prover and verifier engage in the protocol Π_{Hash} with inputs (m, M) and (C_1, C_2) respectively.
4. The prover commits to the signature (r, s) and proves knowledge of an opening. The prover sends $\mathsf{Com}_{pn}(r) = (\mathsf{Com}_p(r), \mathsf{Com}_n(r))$ and $\mathsf{Com}_n(s)$. The prover also commits to the following values: $u_1 = \mathcal{H}(m)s^{-1}, u_2 = rs^{-1}$, and the co-ordinates of the points $u_1 P = (\alpha_x, \alpha_y), u_2 Q = (\beta_x, \beta_y)$, where P is the point of order n in $E(\mathbb{Z}_p)$ used in ECDSA signing, and Q is the ECDSA public key. The prover sends $\mathsf{Com}_n(u_1)$, $\mathsf{Com}_n(u_2)$, $\mathsf{Com}_p(\alpha_x)$,$\mathsf{Com}_p(\alpha_y)$, $\mathsf{Com}_p(\beta_x)$, $\mathsf{Com}_p(\beta_y)$.
5. The prover and the verifier carry out the following Σ-protocol zero-knowledge proofs of knowledge:

(a) $\mathsf{PK}\{(u_1, \alpha_x, \alpha_y, R_1, R_2, R_3) : \mathsf{Com}_p(\alpha_x) = G_1^{\alpha_x} H_1^{R_1} \wedge \mathsf{Com}_p(\alpha_y) = G_1^{\alpha_y} H_1^{R_2} \wedge \mathsf{Com}_n(u_1) = G_2^{u_1} H_1^{R_3} \wedge (\alpha_x, \alpha_y) = u_1 P\}$

(b) $\mathsf{PK}\{(u_2, \beta_x, \beta_y, R_1, R_2, R_3) : \mathsf{Com}_p(\beta_x) = G_1^{\beta_x} H_1^{R_1} \wedge \mathsf{Com}_p(\beta_y) = G_1^{\beta_y} H_1^{R_2} \wedge \mathsf{Com}_n(u_2) = G_2^{u_2} H_1^{R_3} \wedge (\beta_x, \beta_y) = u_2 Q\}$

(c) $\mathsf{PK}\{(r, \alpha_x, \alpha_y, \beta_x, \beta_y, R_1, R_2, R_3, R_4, R_5) : \mathsf{Com}_p(\beta_x) = G_1^{\beta_x} H_1^{R_1} \wedge \mathsf{Com}_p(\beta_y) = G_1^{\beta_y} H_1^{R_2} \wedge \mathsf{Com}_p(\alpha_x) = G_1^{\alpha_x} H_1^{R_3} \wedge \mathsf{Com}_p(\alpha_y) = G_1^{\alpha_y} H_1^{R_4} \wedge \mathsf{Com}_p(r) = G_1^{r} H_1^{R_5} \wedge r = ((\alpha_x, \alpha_y) + (\beta_x, \beta_y))_x\}$

(d) $\mathsf{PK}\{(M, u_1, s, R_1, R_2, R_3) : \mathsf{Com}_n(M) = G_2^{M} H_2^{R_1} \wedge \mathsf{Com}_n(u_1) = G_2^{u_1} H_2^{R_2} \wedge \mathsf{Com}_n(s) = G_2^{s} H_2^{R_3} \wedge M = u_1 s\}$

(e) $\mathsf{PK}\{(r, u_2, s, R_1, R_2, R_3) : \mathsf{Com}_n(r) = G_2^{r} H_2^{R_1} \wedge \mathsf{Com}_n(u_2) = G_2^{u_2} H_2^{R_2} \wedge \mathsf{Com}_n(s) = G_2^{s} H_2^{R_3} \wedge r = u_2 s\}$

6. The prover and verifier engage in Π_{Eq} with input $\mathsf{Com}_{pn}(r)$.

The above protocol can be proven to be a zero knowledge proof of knowledge of ECDSA signature. The proofs for correctness, soundness and zero-knowledge are similar to the proofs of the protocol for the DSA signature.

References

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374. Springer, Heidelberg, March 2008.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796. ACM, 2012.

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 1087–1098. ACM Press, November 2013.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *Advances in Cryptology-Eurocrypt'96*, pages 399–416. Springer, 1996.

[Bra99] Stefan Brands. *Rethinking Public Key Infrastructure and Digital Certificates—Building in Privacy*. PhD thesis, Eindhoven Institute of Technology, Eindhoven, The Netherlands, 1999.

[Cha86] David Chaum. Showing credentials without identification: Signatures transferred between unconditionally unlinkable pseudonyms. In Franz Pichler, editor, *EUROCRYPT'85*, volume 219 of *LNCS*, pages 241–244. Springer, Heidelberg, April 1986.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 1205–1216. ACM Press, November 2014.

[CS97a] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology-CRYPTO'97*, pages 410–424. Springer, 1997.

[CS97b] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424. Springer, Heidelberg, August 1997.

[CZ09] Jan Camenisch and Gregory M Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security*, pages 108–127. Springer, 2009.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *Advances in Cryptology-EUROCRYPT 2000*, pages 418–430. Springer, 2000.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology-ASIACRYPT 2002*, pages 125–142. Springer, 2002.

[DLFKP16] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *IEEE Symposium on Security & Privacy 2016 (Oakland'16)*. IEEE, 2016.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology-EUROCRYPT 2015*, pages 191–219. Springer, 2015.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 16–30. Springer, Heidelberg, August 1997.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in CryptologyCRYPTO86*, pages 186–194. Springer, 1986.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304, 1985.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 171–185. Springer, Heidelberg, August 1987.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 123–128. Springer, Heidelberg, May 1988.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

[ide10] Specification of the identity mixer cryptographic library (revised version 2.3.0). Technical Report RZ 3730, IBM Research, April 2010.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 955–966. ACM Press, November 2013.

[JS07] Stanisław Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology-EUROCRYPT 2007*, pages 97–114. Springer, 2007.

[KKL+16] Vladimir Kolesnikov, Hugo Krawczyk, Yehuda Lindell, Alex J. Malozemoff, and Tal Rabin. Attribute-based key exchange with general policies. Cryptology ePrint Archive, Report 2016/518, 2016. `http://eprint.iacr.org/`.

[Knu69] Donald E Knuth. The art of computer programming. vol. 2: Seminumerical algorithms. addisonwesley. *Reading, MA*, pages 229–279, 1969.

[KS06] Mehmet Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of yaos garbled circuit construction. In *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 *Automata, Languages and Programming*, pages 486–498. Springer, 2008.

[KSS09] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security*, pages 1–20. Springer, 2009.

[Lin15] Yehuda Lindell. An efficient transform from sigma protocols to nizk with a crs and non-programmable random oracle. In *Theory of Cryptography*, pages 93–109. Springer, 2015.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[MR13] Payman Mohassel and Ben Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In *Advances in Cryptology–CRYPTO 2013*, pages 36–53. Springer, 2013.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, pages 275–292, 2005.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in CryptologyCRYPTO91*, pages 129–140. Springer, 1991.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Advances in CryptologyEUROCRYPT96*, pages 387–398. Springer, 1996.

[PZ13] C. Paquin and G. Zaverucha. U-prove cryptographic specification v1.1 (revision 2). Available online: www.microsoft.com/uprove, 2013.

[S+11] Chih-hao Shen et al. Two-output secure computation with malicious adversaries. In *Advances in Cryptology–EUROCRYPT 2011*, pages 386–405. Springer, 2011.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

[Sta96] Markus Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology-EUROCRYPT96*, pages 190–199. Springer, 1996.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Advances in Cryptology-EUROCRYPT 2015*, pages 220–250. Springer, 2015.

Fig. 4. The Protocol $\Pi_{\mathsf{MAC},f}$

Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Eval}, \mathsf{Ve})$ be a garbling scheme. Let F be the following functionality: it takes as inputs x, a, b and outputs v, t such that $v = 1$ if $f(x) = 1$ and 0 otherwise, and $t = ax + b$. The prover has input x, the verifier is in possession of $C_x = \mathsf{Com}(x)$. Both parties have as input the security parameter κ.

1. The verifier generates uniformly random integers a and b of length s and $n + s$ respectively. It commits to them by sending $C_a = \mathsf{Com}(a)$, $C_b = \mathsf{Com}(b)$ and proves knowledge of their opening.
2. The verifier constructs a garbled circuit for F.

$$(GC, e, d) \leftarrow \mathsf{Gb}(1^\kappa, F(x, a, b) = (f(x), ax + b))$$

3. The prover inputs his choice bits by sending (i, x_i) for all $i \in [n]$ to $\mathcal{F}_{COT}$.
4. The verifier inputs the wire keys corresponding to the prover's input by sending (i, K_i^0, K_i^1) for all $i \in [n]$ to $\mathcal{F}_{COT}$.
5. $\mathcal{F}_{COT}$ outputs K_i' for all $i \in [n]$ to the prover where $K_i' = K_i^{x_i}$.
6. The verifier sends the garbled circuit GC to the prover. Note that in what follows, for simplicity, we consider the input keys for a and b to be part of the GC itself, and hence not sent separately.
7. The prover evaluates the garbled circuit

$$(t', Z) \leftarrow \mathsf{Eval}(GC, \{K_i'\}_{i \in [n]})$$

8. Prover commits to the garbled output Z by sending $\mathsf{Com}(Z)$ to the verifier and proves knowledge of opening.
9. Verifier sends the decoding information d_t for t.
10. Prover decodes

$$t = \mathsf{De}(d_t, t')$$

and commits to the decoded output by sending $C_t = \mathsf{Com}(t)$, and proves knowledge of opening.
11. Verifier sends open to $\mathcal{F}_{COT}$.
12. $\mathcal{F}_{COT}$ sends (K_i^0, K_i^1) to the prover for all $i \in [n]$.
13. Verifier opens $\mathsf{Com}(a)$ and $\mathsf{Com}(b)$. Prover checks the openings and aborts if they fail.
14. Prover verifies that the correct circuit was garbled by running $\mathsf{Ve}(GC, \{K_i^0, K_i^1\}_{i \in [n]}, F)$. It also checks that garbled inputs for x, a, b are the correct one. If any of checks fail, the prover terminates. Otherwise, it receives the decoding vector d, and he opens the commitment to the output Z by sending Z and randomness.
15. Verifier checks that the opening is correct and that $\mathsf{De}(d, Z) = 1$. If the opening is not correct or if $\mathsf{De}(d, Z) \neq 1$, the verifier outputs reject and terminates.
16. If the verifier did not terminate, the prover and the verifier engage in a Zero-knowledge protocol to prove the following:

$$\mathsf{PK}\{(x, t, r, R) : C_x = \mathsf{Com}(x) \wedge C_t = \mathsf{Com}(t) \wedge t = ax + b\}$$

17. If the zero-knowledge proof verifies, the verifier outputs accept.