Efficient Non-Interactive Zero-Knowledge Proofs in Cross-Domains without Trusted Setup*

Michael Backes[1], Lucjan Hanzlik[2], Amir Herzberg[3], Aniket Kate[4], and Ivan Pryvalov (✉)[1]

[1] CISPA Helmholtz Center for Information Security
backes@cispa.saarland, ivan.pryvalov@cispa.saarland
[2] Stanford University, CISPA Helmholtz Center for Information Security
lucjan.hanzlik@stanford.edu
[3] University of Connecticut, Bar Ilan University
amir.herzberg@uconn.edu
[4] Purdue University
aniket@purdue.edu

Abstract. With the recent emergence of efficient zero-knowledge (ZK) proofs for general circuits, while efficient zero-knowledge proofs of algebraic statements have existed for decades, a natural challenge arose to combine algebraic and non-algebraic statements. Chase et al. (CRYPTO 2016) proposed an interactive ZK proof system for this cross-domain problem. As a use case they show that their system can be used to prove knowledge of a RSA/DSA signature on a message m with respect to a publicly known Pedersen commitment $g^m h^r$. One drawback of their system is that it requires interaction between the prover and the verifier. This is due to the interactive nature of garbled circuits, which are used in their construction. Subsequently, Agrawal et al. (CRYPTO 2018) proposed an efficient non-interactive ZK (NIZK) proof system for cross-domains based on SNARKs, which however require a trusted setup assumption.

In this paper, we propose a NIZK proof system for cross-domains that requires no trusted setup and is efficient both for the prover and the verifier. Our system constitutes a combination of Schnorr based ZK proofs and ZK proofs for general circuits by Giacomelli et al. (USENIX 2016). The proof size and the running time of our system are comparable to the approach by Chase et al. Compared to Bulletproofs (SP 2018), a recent NIZK proofs system on committed inputs, our techniques achieve asymptotically better performance on prover and verifier, thus presenting a different trade-off between the proof size and the running time.

1 Introduction

Zero-knowledge (ZK) proofs, introduced by Goldwasser, Micali, and Rackoff [25], are one of the central cryptographic building blocks, which allow a prover to convince a verifier that a statement is true without revealing any other information.

* Accepted to PKC 2019. This is the submitted version.

Goldreich, Micali, and Wigderson showed that ZK proofs for NP-languages are possible [24], which opened up a number of new research directions in cryptography.

Zero-knowledge proof systems are an essential building block used in many privacy-preserving systems, e.g. anonymous credential systems [12] and voting protocols [26,7]. Unfortunately, only a few systems have been used in practice. The main reason is that ZK proofs for general statements are usually inefficient. Thus, the research focus switched from general statements to interesting subclasses. In particular, a prover can efficiently prove knowledge of discrete logarithms in groups of known [18,34] and unknown [14,6] order. Those proofs were extended to allow other statements, e.g., equivalence of discrete logarithms, or knowledge of representation. The main advantage was that using the Fiat-Shamir transformation [21] one can make those systems non-interactive (NIZK), i.e. no interaction between the prover and the verifier is necessary to generate the proof, and transform honest-verifier ZK protocols into full ZK. Groth and Sahai [28,20] further extended the class of efficient NIZK proofs to statements about pairing product equations. The common factor of those proofs is that they are restricted to algebraic groups and cannot be efficiently used to prove statements about non-algebraic structures, e.g., the SHA hash function or the AES encryption scheme.

The problem of efficient interactive ZK proofs for non-algebraic statements was solved by Jawurek et al. [30]. Their system allows to efficiently prove statements of the following form: "The prover knows an input x such that $y = SHA(x)$ for some public y". Unfortunately, one cannot apply the Fiat-Shamir transformation to make those proofs non-interactive. The system is based on garbled circuits [37], which are private coin protocols, which in turn makes the system inherently interactive. Giacomelli et al. [23] addressed this limitation and introduced ZKBoo, a non-interactive proof system for arithmetic circuits, based on the "MPC-in-the-head" technique [29]. In their system, the proof size depends linearly on the number of gates, input and output wires. This work was further improved by Chase et al. [16] with the introduction of the ZKB++ system. The authors were able to reduce the proof size by a constant factor and addressed post-quantum security of the construction. Ames et al. [4] proposed Ligero, a NIZK proof system based on the "MPC-in-the-head" technique, which has the proof size proportional to the square root of the verification circuit size.

An interesting line of research present succinct non-interactive zero-knowledge proofs (SNARKs) [27,22,9]. They allow compact proofs and very efficient verification, but require a complex trusted setup and the prover has to perform a number of public key operations (i.e. modular exponentiations or equivalently elliptic curve point multiplications) proportional to the circuit size. The setup algorithm can be executed by a trusted party or by the participants of the system using multi-party computation (MPC).

While there exist efficient proofs for algebraic and non-algebraic statements, it became a natural challenge to combine both worlds and create a proof system that would work efficiently in both, algebraic and non-algebraic, domains.

Obviously, one can implement algebraic structure directly using non-algebraic statements by defining all group operations as functions. This approach introduces a significant overhead in size of the proven statement, which increases the size of the proof and the number of required computations. As noted by [3], depending on the group size, the circuit for computing a single exponentiation could be *thousands or millions* of gates. Alternatively, one can implement arithmetic circuit directly using algebraic statements by treating each gate in a circuit as an algebraic function and proving relations between gates. The prover's/verifier's work and the proof size would be linear in the number of gates, and in case of hash functions or block ciphers it could be *tens of thousands* of public key operations and group elements.

The first attempt to efficiently solve this cross-domain problem was the Crypto'16 work by Chase et al. [17]. Their system can be used e.g. to prove that a given algebraic commitment (e.g. Pedersen commitment) C is a commitment to x, where $F(x) = 1$ and F is expressed as a boolean circuit. The authors show that an efficient proof system for this statement can be used as a building block to construct more efficient proofs of knowledge of a signature and a committed message for RSA-FDH, DSA and EC-DSA signatures. Their system can be extended to a scenario, where we have k commitments to $x_1, \ldots, x_k$ and the input x is the concatenation $x_1 || \ldots || x_k$ of values in those commitments.

Chase et al. propose two constructions of their proof system. For the first the number of public key operations is linear in the size of x and that of symmetric key operations is proportional to the number F_g of gates in F. The second construction reduced the number of public key operations to a number linear in the security parameter λ, but this comes at a cost of additional symmetric key operations which are proportional to $F_g + |x| \cdot \lambda$. Unfortunately, their approach is based on the ZK proofs from [30] and the proof system is therefore interactive by nature.

Bünz et al. [11] presented at S&P'18 efficient NIZK range proofs called Bulletproofs. Those proofs can also be used for proving statements expressed as arithmetic circuits with algebraically committed inputs. The proof technique relies on discrete log assumptions and the Fiat-Shamir transformation. While Bulletproofs produce relatively short proofs, the prover's work is still expensive, specifically, the prover has to perform a number of public key operations linear in the circuit size.

At Crypto'18, Agrawal, Ganesh, and Mohassel [3] presented non-iteractive zero-knowledge proofs for composite statements. Whereas the authors addressed the same problem of constructing zero-knowledge proofs in cross-domains, theirs and our proposals differ in the underlying cryptographic blocks that handle the arithmetic part of the proof system. Specifically, their proofs are based on Σ-protocols and SNARKs [22]. As already noted, SNARKs allow for short proofs and fast verification of arithmetic statements, however they require a trusted setup for generating the common reference string (CRS) for a particular circuit F. Typically, the CRS needs to be regenerated for a different circuit F'. This

is not desirable in some applications such as ZCash, where an expensive MPC protocol has to be run to generate a CRS [2].

Our contribution. In this work, we present an efficient (both for the prover and the verifier) non-interactive zero-knowledge proof system for cross-domains that requires no trusted setup assumption. Our system uses ZKB++ [16] as a building block, which is based on a technique called "MPC-in-the-head" [29]. The idea is that the prover represents the circuit F as a multi-party computation (MPC) and generates three shares $x_1 \oplus x_2 \oplus x_3 = x$, where x is the original input of the prover. The prover then performs the MPC computation using the values x_1, x_2, x_3 and given a challenge $e \in \{1, 2, 3\}$ returns the view of computations performed with inputs x_e and x_{e+1}. Executing these steps a number of times decreases the soundness error of the proof. What is more, we can apply the Fiat-Shamir transformation and allow the prover to compute this challenge itself, making the system non-interactive.

We extend this idea to allow algebraic statements. To illustrate our solution let us consider a simple example where the prover publishes $y = SHA(x)$ and a Pedersen commitment $C = g^x \cdot h^r$. In this case, the prover wants to convince the verifier that he knows x. To do so, he performs the "MPC-in-the-head" as in ZKB++ and computes Pedersen commitments to all bits of the values x_1, x_2, x_3. Upon receiving a challenge $e \in \{1, 2, 3\}$, additionally to the views of the MPC the prover opens all commitments to the bits of x_e and x_{e+1}. Finally, to bind the "MPC-in-the-head" part to the Pedersen commitment C, the prover computes commitments to the bits of the value $x_e \oplus x_{e+1} \oplus x_{e+2}$ and proves that these commitments contain the binary representation of the same value that is in C. As in ZKB++, this extended system has to be executed several times in order to decrease the probability of the prover cheating the verifier. However, in contrast to [17] we can apply the Fiat-Shamir transformation to get a NIZK system.

The number of public key operations in our system is proportional to $|x| \cdot \lambda$. This follows directly from the way we combine both domains. Each round we have to prove that the commitments to the bits of $x_e \oplus x_{e+1} \oplus x_{e+2}$ are the binary representation of x. We solve this obstacle by committing to full values of the ZKB++ share and not to its bits and show that we can still compute the XOR value of them because 2 out of 3 values are revealed by the ZKB++ protocol. This unique technique allows us to further decrease the number of public key operations to $O(|x| + \lambda)$.

The contribution of this paper can be summarized as follows. We are the first to present an efficient (both for the prover and the verifier) non-interactive zero-knowledge (NIZK) proof system for algebraic and non-algebraic domains (cross-domains) that requires no trusted setup. The solution is based on a combination of ZKBoo [23,16] with standard Schnorr based proofs [34,15]. Using our techniques, we obtain the efficient non-interactive proof of knowledge (proof of possession) of DSA/RSA signatures, without revealing the signature itself.

Applications. A straightforward application of zero-knowledge proofs in cross-domains are anonymous credentials. Chase et al. [17] observed that many existing

credential systems [10,12,13,8,5] rely on signature schemes that are tailored in a specific way to provide the desired properties of the system. The user proves that he knows a value x and a signature under this value. Using zero-knowledge proofs in cross-domains allows to use standard signature schemes like RSA-FDH or DSA for which there exist no efficient proof system in the standard algebraic setting. In contrast to the system by Chase et al. our proofs are non-interactive, which means that they can be used to construct round-optimal anonymous credential systems. This implies that using our techniques, one can create concurrently secure systems based on RSA and DSA signatures.

Another application of NIZK in cross-domains, mentioned in [3], are proofs of solvency for Bitcoin exchanges. In this scenario, an exchange wants to prove to its customers that it is solvent, i.e. that it has enough Bitcoins to cover its liabilities. To this end, the exchange would need to prove the control over some Bitcoin addresses. A certain Bitcoin address is a 160-bit hash of a public ECDSA key [1]. The corresponding proof is a proof of knowledge x such that $H(g^x) = y$, where H is a hash-function such as SHA-256. Here, only y is public, and the exchange would like to keep its public key part g^x hidden, otherwise an adversary could track the movement of exchanges associated with its public key. Since the Bitcoin network does not require a trusted setup assumption, proofs of solvency based on the approach by [3] would require a trusted CRS generation to be done. On the other hand, since our techniques do not require any trusted setup assumption, they can be used directly to prove solvency for Bitcoin exchanges. The proof system would additionally include a proof of equality of discrete logarithm of a committed value and another committed value. More specifically, a prover would need to prove knowledge of x such that $H(g^x) = y$ for some public y. Here the input to the circuit H is g^x. The prover has to commit to g^x as Com_{g^x} and to x as Com_x and use the proof of equality of discrete logarithm of a committed value and another committed value, for which we refer to [17].

Note that ours and the proof system by Chase et al. [17] or any other system cannot be post-quantum secure if the underlying security assumptions in the algebraic domain (integer factorization, discrete logarithms) can be broken by a quantum adversary [35].

Comparison with existing techniques. We compare ours and prior work on zero-knowledge proofs in cross-domains in Table 1. We discuss the efficiency of the constructions based on a circuit F and a committed input x. For the algebraic part of the proof system, Σ-protocols are used in all ZK proof systems presented in the table. Σ-protocols require a constant number of public-key operations for a single algebraic statement and do not require any trusted setup assumption. The approach by Chase et al. [17] is the only interactive protocol in the table. In their first construction, the arithmetic part of the proof system is based on garbled circuits, whose prover's/verifier's cost amounts to $O(|F|)$ of symmetric-key operations. The number of public key operations is linear in the input size $|x|$. In the second construction, Chase et al. achieve the number of public key operations independent of $|x|$ at the cost of increasing the circuit that has to be garbled. Various techniques to reduce computation, communication, memory

requirements of garbled circuits are available, e.g. [32,38,36]; in [31] XOR-gates can be garbled essentially at no cost. In Bulletproofs [11], the prover has to perform a constant number of public key operations for each multiplication gate of the circuit, while the verifier is more efficient due to the multi-exponentiation trick. The proof size in Bulletproofs is logarithmic in the number of multiplication gates in the arithmetic circuit for verifying the witness. The approach by Agrawal et al. [3], which is based on SNARKs, is the only protocol that requires a trusted setup assumption and produces constant proofs. Verifier's work does not depend on the circuit size, and the number of public key operations is linear in the input size. Prover's work requires a number of public key operations linear in the circuit size. We analyze efficiency of our Construction 2. As we show in Section 3.4, it requires $O(|x| + \lambda)$ public key operations, while the number of symmetric-key operations is $O(|F| \cdot \lambda)$, since ZKB++ protocol has to be repeated to reduce the soundness error to a negligible value. Proof size is dominated by ZKB++ and amounts to $O((|F|\lambda + |x|)\lambda)$.

Table 1: Comparison of ZK proof systems in cross-domains for a circuit F with an algebraically committed input x, where $|F|$ denotes the circuit size, $|x|$ the number of input bits. We denote by λ the security parameter, by pub a public-key operation, by sym a symmetric-key operation.

	Non-interactive	Without trusted setup	Prover's work	Verifier's work	Communication/ Proof size
CGM16 [17] Constr.1	No	Yes	$O(\|x\| \cdot \text{pub} + \|F\| \cdot \text{sym})$	$O(\|x\| \cdot \text{pub} + \|F\| \cdot \text{sym})$	$O((\|F\| + \|x\|)\lambda)$
CGM16 [17] Constr.2	No	Yes	$O(\lambda \cdot \text{pub} + (\|F\| + \|x\|\lambda) \cdot \text{sym})$	$O(\lambda \cdot \text{pub} + (\|F\| + \|x\|\lambda) \cdot \text{sym})$	$O((\|F\| + \|x\|\lambda)\lambda)$
BBB+18 [11]	Yes	Yes	$O(\|F\| \cdot \text{pub})$	$O(\frac{\|F\|}{\log(\|F\|)} \cdot \text{pub})$	$O(\log(\|F\|)\lambda)$
AGM18 [3]	Yes	No	$O((\|F\| + \lambda) \cdot \text{pub})$	$O((\|x\| + \lambda) \cdot \text{pub})$	$O(\lambda)$
This work	Yes	Yes	$O((\|x\| + \lambda) \cdot \text{pub} + (\|F\| \cdot \lambda) \cdot \text{sym})$	$O((\|x\| + \lambda) \cdot \text{pub} + (\|F\| \cdot \lambda) \cdot \text{sym})$	$O((\|F\|\lambda + \|x\|)\lambda)$

Paper Outline. The rest of the paper is organized as follows. Section 2 contains preliminaries. In Section 3, we develop our solution for NIZK proofs in cross-domains. The section starts with the problem statement for NIZK proofs in cross-domains. Then, in Section 3.1 we present our first attempt cross-domain NIZK proof system based on ZKB++ followed by its security analysis. Next, in Section 3.2 we present an improved version and its security analysis. Finally, in Section 3.3 we describe the optimization technique to reduce the number of public key operations and in Section 3.4 we perform efficiency analysis of our constructions. In Section 4, we complement NIZK proofs in cross-domains to allow OR-proofs. Section 5 concludes.

2 Preliminaries

In this section we recall the notions of commitment schemes, zero-knowledge and Σ-protocols. We also recall the details of the ZKBoo protocol introduced by Giacomelli et al. [23].

2.1 Homomorphic Commitment Schemes

Let us by $\mathcal{M}_{\mathsf{ck}}$ denote the message space of the commitment scheme and by $\mathcal{OI}_{\mathsf{ck}}$ the space of opening information (also called randomness).

Definition 1 (Commitment Scheme). *A commitment scheme consists of the following PPT algorithms* $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$*:*

$\mathsf{Gen}(\lambda)$*: on input security parameter* λ*, this algorithm outputs a commitment key* ck*, which is an implicit input for the below algorithms.*

$\mathsf{Com}(m, r)$*: on input message* m *and opening information* r*, this deterministic algorithm outputs a commitment* C_m*.*

$\mathsf{Open}(C_m, m, r)$*: on input commitment* C_m*, message* m *and opening information* r*, this algorithm outputs a bit* $\{0, 1\}$*.*

Definition 2 (Perfect Hiding). *A commitment scheme is perfectly hiding, if for all adversaries* $\mathcal{A}$ *we have:*

$$\Pr[\mathsf{ck} \leftarrow \mathsf{Gen}(\lambda), (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{ck}), r \xleftarrow{\$} \mathcal{OI}_{\mathsf{ck}}, C \leftarrow \mathsf{Com}(m_0, r) : \mathcal{A}(\mathsf{st}, C) = 1] =$$
$$\Pr[\mathsf{ck} \leftarrow \mathsf{Gen}(\lambda), (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{ck}), r \xleftarrow{\$} \mathcal{OI}_{\mathsf{ck}}, C \leftarrow \mathsf{Com}(m_1, r) : \mathcal{A}(\mathsf{st}, C) = 1].$$

Definition 3 (Computational Binding). *A commitment scheme is computationally binding, if for all PPT adversaries* $\mathcal{A}$ *we have:*

$$\begin{aligned} |\Pr[(\mathsf{ck}) \leftarrow \mathsf{Gen}(\lambda), (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(\mathsf{ck}) : m_0 \neq m_1 \ \wedge \\ \mathsf{Com}(m_0, r_0) = \mathsf{Com}(m_1, r_1)]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{binding}}(\lambda), \end{aligned}$$

where we require that $m_0, m_1 \in \mathcal{M}_{\mathsf{ck}}$*,* $r_0, r_1 \in \mathcal{OI}_{\mathsf{ck}}$ *and* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{binding}}(\lambda)$ *is negligible in the security parameter* λ*.*

Definition 4 (Equivocality). *A commitment scheme is equivocal, if there exists an algorithm* Eval *and an alternative* Gen' *algorithm that additionally to the commitment key* ck *returns a trapdoor* τ *such that given a commitment* $C = \mathsf{Com}(m, r)$ *we have* $C = \mathsf{Com}(m', \mathsf{Eval}(\tau, m', (C, m, r)))$ *for any message* m'*, i.e.* Eval *can be used to compute the randomness to open* C *to an arbitrary value.*

Moreover, we assume that there exists an efficient extraction algorithm $\mathsf{Extr}_{\mathsf{ck}}$ *that given two openings of the same commitment, i.e.* (m_1, r_1, m_2, r_2) *where* $\mathsf{Com}(m_1, r_1) = \mathsf{Com}(m_2, r_2)$ *and* $m_1 \neq m_2$*, returns the trapdoor* τ*.*

We require that a commitment scheme is binding, hiding and equivocal. Additionally, in this paper we assume that the used commitment scheme has the following homomorphic property: for all $m_1, m_2 \in \mathcal{M}_{\mathsf{ck}}$ and $r_1, r_2 \in \mathcal{OI}_{\mathsf{ck}}$ we have: $\mathsf{Com}(m_1, r_1) \cdot \mathsf{Com}(m_2, r_2) = \mathsf{Com}(m_3, r_3)$, where $m_3 = m_1 + m_2$ and $r_3 = r_1 + r_2$. This homomorphism allows us to introduce multiplication by a known scalar, i.e. given $C = \mathsf{Com}(m, r)$ we can compute $C' = [k]C = \mathsf{Com}(k \cdot m, k \cdot r)$, where by $[k]C$ we denote multiplication of commitment C by a public scalar k. What is more, for a given commitment $C_b = \mathsf{Com}(b, r)$ to a bit b, we can easily compute the exclusive-or on this hidden value with a known bit α. If $\alpha = 0$, we leave C unchanged, otherwise, if $\alpha = 1$, we compute $C_{b \oplus \alpha} = C_1 / C_b = \mathsf{Com}(1 - b, -r)$, where $C_1 = \mathsf{Com}(1, 0)$ is formally a commitment to 1 with no randomness (instead of sampling it, it is set to 0). Note that for commitments $C_x = \mathsf{Com}(x, r) = \prod_{i \in [0, |x|-1]} [2^i] C_{x[i]} = \prod_{i \in [0, |x|-1]} [2^i] \mathsf{Com}(x[i], r_{x[i]})$, where $x[i]$ is the i-th bit of x, we can compute a commitment $C_{x \oplus \alpha}$ for a known α. To do so, we apply the above technique bitwise, i.e. to commitments $C_{x[i]}$ and using the new values we then compute the commitment $C_{x \oplus \alpha}$. Notice, that this operation changes the opening information, which is now $\sum_{i \in [0, |x|-1]} (-1)^{\alpha[i]} r_{x[i]}$, i.e. $C_{x \oplus \alpha} = \mathsf{Com}(x \oplus \alpha, \sum_{i \in [0, |x|-1]} (-1)^{\alpha[i]} r_{x[i]})$.

An example of a scheme that has those properties is the one introduced by Pedersen [33]. There, given a commitment key $\mathsf{ck} = (g, h, q, p)$, a message $m \in \mathbb{Z}_q$ and an opening information $r \in \mathbb{Z}_q^*$ the commitment is of the form $\mathsf{Com}(m, r) = g^m \cdot h^r \mod p$. Multiplying two commitments $\mathsf{Com}(m_1, r_1) \cdot \mathsf{Com}(m_2, r_2)$ we get $g^{m_1 + m_2} h^{r_1 + r_2}$, which is a commitment to message $m_1 + m_2 \mod q$ with opening information $r_1 + r_2 \mod q$, as required. Note that the commitment scheme is also equivocal and that there exists an extraction algorithm $\mathsf{Extr}_{\mathsf{ck}}$ (to this end, one simply needs to choose public parameters g, h with a known discrete logarithm).

2.2 Zero-Knowledge Proofs

Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ be an efficiently computable binary relation, for which $\mathcal{R}(x, w) = 1 \iff (x, w) \in \mathcal{R}$. We call x a statement and w a witness. A very simple example of such a relation is $\mathcal{R} = \{(x, w) : x = SHA(w)\}$, where we are given a SHA value as part of the statement and the preimage is part of the witness. Obviously, given both values we can easily verify that $\mathcal{R}(x, w) = 1$ by computing the SHA value on w and comparing it with x. We will assume that $|w| \leq \mathsf{poly}(|x|)$, which means that the witness length should be polynomial in the statement length. We will denote by $L_{\mathcal{R}}$ the language consisting of true statements in $\mathcal{R}$, i.e. $L_{\mathcal{R}} = \{x | \exists w : (x, w) \in \mathcal{R}\}$.

We call a cryptographic protocol between two PPT parties, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ an *argument* for language $L_{\mathcal{R}}$ if it has the following properties. Using communication $\mathcal{P}$ wants to convince $\mathcal{V}$ that $x \in L_{\mathcal{R}}$, where x is a publicly known statement. Obviously, the prover has some extra private input, e.g. he knows a witness for which $\mathcal{R}(x, w) = 1$.

At the end of the protocol the verifier outputs accept if he is convinced and reject otherwise. The protocol is *complete* if for all $x \in L_{\mathcal{R}}$ an honest prover

always convinces an honest verifier. We also require that if $x \notin L_{\mathcal{R}}$, then a cheating prover has only a small chance ϵ (called *soundess error*) to convince an honest verifier. This property should hold for all possible statements not in the language, i.e. for all $x \notin L_{\mathcal{R}}$ we have $\Pr[\mathcal{V}(x) = \mathsf{accept}] \leq \epsilon$. Finally, we require a property called *zero-knowledge* (ZK). Informally, this means that whatever strategy a verifier follows, he learns nothing besides whether $x \in L_{\mathcal{R}}$. It follows that he cannot get any information about the private input of the prover. A weaker notion of ZK is called honest-verifier ZK (HVZK). Zero-knowledge property in this case holds only for a verifier, who does not deviate from the protocol.

A special case of such arguments are Σ-protocols, which follow a specific communication pattern similar to the letter Σ. In the rest of the paper we will only consider this type of protocols.

Definition 5 (Σ-Protocol). *A protocol $\Pi_{\mathcal{R}}$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ is a Σ-protocol for relation $\mathcal{R}$ if:*

- *The protocol consists of three phases:*
  1. *(Commit) $\mathcal{P}$ sends a message a to $\mathcal{V}$,*
  2. *(Challenge) $\mathcal{V}$ picks a random e and sends it to $\mathcal{P}$,*
  3. *(Response) $\mathcal{P}$ sends a second message z to $\mathcal{V}$.*
- *$\Pi_{\mathcal{R}}$ is complete - if both parties are honest, then for all $x \in L_{\mathcal{R}}$ we have $\Pr[(\mathcal{P}, \mathcal{V})(x) = 1] = 1$.*
- *$\Pi_{\mathcal{R}}$ is s-special sound - for any x and any set of s accepting conversations $T = \{(a, e_i, z_i)\}_{i \in \{1,\ldots,s\}}$, where $e_i \neq e_j$ if $i \neq j$, there exists an efficient algorithm Extr that on input T outputs w such that $\mathcal{R}(x, w) = 1$.*
- *$\Pi_{\mathcal{R}}$ is a special honest-verifier ZK (HVZK) - there exists a PPT simulator $\mathcal{SIM}$ such that on input $x \in L_{\mathcal{R}}$ outputs a triple (a', e, z') with the same probability distribution of real conversations (a, e, z) of the protocol.*

The last property ensures only that Σ-protocols are ZK if the verifier is honest and does not base his challenge e on the first message of the prover. Σ-protocols have found many applications in the design of efficient identification and signature schemes. The main advantage of using those protocols is that using the Fiat-Shamir transformation [21], they can be made non-interactive in the random oracle model. What is more, using this technique the protocol is ZK even if the verifier is dishonest. Note that if the challenge e is chosen from a set of cardinality c, then s-special soundness implies that the soudness error is $(s-1)/c$.

Notation. Given two commitments $C_x = \mathsf{Com}(x, r_x)$ and $C_y = \mathsf{Com}(y, r_y)$ we will denote by $\mathcal{P}\{(C_x \equiv C_y)\}$ the prover's part and by $\mathcal{V}\{(C_x \equiv C_y)\}$ the verifier's part of a Σ-protocol, where the prover tries to convince the verifier that it knows openings (x, r_x) and (y, r_y) of public commitments C_x and C_y, respectively, such that $x = y$. There exist very efficient Σ-protocols for the above mentioned Pedersen commitments. In such a case, the witness is composed of the committed value x and the opening informations r_x and r_y. We may sometimes append the notation to denote a subroutine algorithm such as *Commit*,

Response, or *Reconstruct*. The *Commit* subroutine has a special output notation. We denote by (st, a) the result of *Commit* execution, where st denotes an internal state and a the output.

Notation for a bit commitment. We will also use this notion for a special case, where the prover wants to show that the value committed in $C_x = \mathsf{Com}(x, r_x)$ is a bit, i.e. $x \in \{0, 1\}$. We will use $\mathcal{P}\{(C_x \equiv C_0) \vee (C_x \equiv C_1)\}$ to denote this special case. Note that we do not necessarily require the use of commitments to values 0 (C_0) and 1 (C_1), as there exist more efficient realizations, i.e. given a commitment $C = g^x \cdot h^r$ the prover simply shows that it knows the discrete logarithm of C or C/g to the base of h, and therefore C_0 and C_1 may be omitted. Moreover, we will use $\prod_{i=0}^{|x|-1} C_{2^i \cdot x[i]} = \mathsf{Com}(x, r)$ to denote a commitment to x, where $x = \sum_{i=0}^{|x|-1} 2^i x[i]$, $r = \sum_{i=0}^{|x|-1} 2^i r_{x[i]}$, and $C_{x[i]} = \mathsf{Com}(x[i], r_{x[i]})$.

2.3 ZKBoo/ZKB++

Giacomelli et al. [23] proposed ZKBoo, an efficient Σ-protocol based on the idea "MPC-in-the-head" [29]. Subsequently, Chase et al. [16] presented ZKB++, the successor of ZKBoo, which has more compact proofs. As both versions of the protocols differ primarily in technical aspects, our techniques can be applied to either version. The main advantage of this system over the one by Jawurek et al. [30] is that it can be made non-interactive using the Fiat-Shamir transformation.

ZKBoo/ZKB++ work for arithmetic functions F with prover's input x and the verifier holding no private input. Let y denote the output of function, i.e. $y = F(x)$. To create such a zero-knowledge proof of x, the prover splits the input into 3 shares (x_1, x_2, x_3) and for each pair x_i, x_{i+1} runs the function $F'(x_i, x_{i+1})$ to obtain y_i. F' is constructed in such a way that the correctness property of ZKBoo/ZKB++ ensures $y_1 \oplus y_2 \oplus y_3 = y$. The prover commits to all three views. The verifier sends a challenge $e \in \{1, 2, 3\}$, which can be replaced by the output of the random oracle applied on appropriate inputs. The prover opens input shares (x_e, x_{e+1}) and the randomness used in computing $F'(x_e, x_{e+1})$ in the corresponding two views. The verifier then checks whether y_e was correctly computed or not. Another property of F' is that two out of three views leak no information about the input x (the property is called 2-privacy; for more details we refer to Definition 3 in [23]). The protocol is 3-special sound and the soundness error of the protocol is 2/3. Therefore, to reduce the soundness error to a negligible value the prover runs multiple independent rounds of ZKBoo/ZKB++ protocol. In Fig. 5 in Appendix we present the non-interactive version of the ZKB++ protocol.

3 Combining ZKB++ with Algebraic Commitments

In this section we present our main contribution: a Σ-protocol for statements in cross-domains. Throughout this paper we will consider the following statement

as the main building block that can be composed to create proofs for more general statements.

Statement 1 *Prove that there exists x such that $F(x) = 1$ and x is committed to C_x, where x is a $|x|$-bit number, F is an arithmetic circuit, and the commitment scheme is based on the group structure of order larger than $2^{|x|}$ and allows some homomorphic operations.*

The naive approach to prove this statement is just to implement all algebraic operations as a part of the circuit F and execute the ZKB++ protocol. However, Chase et al. [17] already noticed that expressing modular exponentiation in a boolean circuit would be computationally too expensive and fairly inefficient. In particular, since the number of gates increases non-linearly in the size of the input, this also means that the proof size increases at the same rate and so does the time required to compute the proof. As we will show, there exists a more efficient way of realizing this kind of proofs.

3.1 Our Technique - First Approach

We propose the following technique, in which we take advantage of:

1) the fact that the ZKB++ protocol is a Σ-protocol,
2) the additive sharing of the prover's input x in the group $\mathbb{Z}_2$ in ZKB++,
3) that Σ-protocols can be executed in parallel,
4) a multiplicatively homomorphic commitment scheme in the group $\mathbb{Z}_q$; for simplicity we assume that $2^{|x|} < q$, the other case is addressed in Section 3.3.

The overall idea is to combine a ZKB++ round with zero-knowledge proofs that input bits of x are bound to the public commitment. This part involves ZK proofs for all individual bits of the ZKB++ input and the three exclusive-or based bit shares. In particular, we prove that the exclusive-or value of those shares is given in a commitment and is equal to the bit representation of x. We then prove that values in the commitments match the real shares by giving opening information for 2 out of 3 commitments, depending on the shares revealed by ZKB++. More details can be found in Construction 1.

Construction 1 (Cross-ZKB++ First Attempt) *Let $x[i]$ denote the i-th bit of input x, i.e. $x = (\ldots, x[1], x[0])$. In the following, we describe necessary steps to add to the ZKB++ protocol (Fig. 5) in order to realize the connection between the input bits $(\ldots, x[1], x[0])$ of the function F and the public commitment C_x, as defined in Statement 1.*

- *(Commit Phase) — The prover follows the steps specified by the ZKB++ protocol. Then, for each bit i of input x the prover commits to $x[i]$ and to the respective input shares $x[i]_1, x[i]_2, x[i]_3$ and gets $C_{x[i]}, C_{x[i]_1}, C_{x[i]_2}, C_{x[i]_3}$.*

Again, for each bit i the prover executes the commit phase of a Σ-protocol (with challenge space $\{1,2,3\}$) for the following algebraic statement:

$$\mathcal{P}\{\big((C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\big) \wedge \big(\prod_{i\in[0,|x|-1]} C_{2^i \cdot x[i]} \equiv C_x\big) \wedge \big((C_{x[i]_1} \cdot C_{x[i]_2} \cdot C_{x[i]_3} \equiv C_{x[i]}) \vee (C_{x[i]_1} \cdot C_{x[i]_2} \cdot C_{x[i]_3} \equiv C_{2+x[i]})\big)\}. \quad (1)$$

The prover sends commitments $\{C_{x[i]}, C_{x[i]_1}, C_{x[i]_2}, C_{x[i]_3}\}_{i\in[0,|x|-1]}$, and the commitments from the ZKB++ protocol and the Σ-protocol to prove the statement Eq. (1) to the verifier.

- *(Challenge Phase) — The verifier sends the challenge $e \in \{1,2,3\}$.*
- *(Response Phase) — The prover executes the last phase of the ZKB++ and the other proofs, and sends the result to the verifier. Additionally, he sends the opening information for commitments $C_{x[i]_e}$, $C_{x[i]_{e+1}}$, for all $i \in [0, |x| - 1]$.*

To verify the result the verifier follows the steps specified by the ZKB++ protocol and additionally performs the following checks: reject if the opening is wrong or the bits of the shares don't match the bits in the ZKB++ views, or if any of the additional algebraic proofs is invalid.

We present in Figs. 1 and 2 the detailed description of Construction 1, instantiated with t rounds of ZKB++ and made non-interactive using the Fiat-Shamir transformation.

Note that the proof system Eq. (1) does not explicitly enforce $C_{x[i]_1}, C_{x[i]_2}, C_{x[i]_3}$ to be commitments to bits. However, as we show in the proof of Theorem 1, it is the case.

Security analysis

Lemma 1. *Assuming the ZKB++ protocol is complete, the Σ-protocols for the algebraic statements are complete and the used commitment scheme is homomorphic, then Construction 1 is a complete Σ-protocol for the statement in Problem 1.*

Proof. Follows by inspection. □

Theorem 1. *Assuming the ZKB++ protocol is 3-special sound, the Σ-protocols for the algebraic statements are 2-special sound and the used commitment scheme is homomorphic and equivocal, then Construction 1 is a 3-special sound Σ-protocol for Statement 1.*

Proof. We will prove this theorem by constructing an efficient algorithm Extr_{Cross} that using 3 accepting tuples (a, e_1, z_1), (a, e_2, z_2) and (a, e_3, z_3) can compute $w^* = (x^*, r^*)$, such that $F(x^*) = 1$ and $C_x = \mathsf{Com}(x^*, r^*)$, which is a valid witness for the proven statement.

The algorithm works as follows:

$p \leftarrow \mathsf{Prove}(x, C_x = \mathsf{Com}(x, r))$

// (*Commit* step)
$(st_\zeta, a_\zeta) \leftarrow \mathsf{ZKB}_F.Commit(x)$
foreach $i \in [0, |x| - 1]$ **do**
  $C_{x[i]} = \mathsf{Com}(x[i], r_i)$
  foreach $\rho \in [1, t]$ **do**
    Extract shares $x[i]_1^{(\rho)}, x[i]_2^{(\rho)}, x[i]_3^{(\rho)}$ from st_ζ
    $C_{x[i]_1^{(\rho)}} = \mathsf{Com}(x[i]_1^{(\rho)}, r_{x[i]_1^{(\rho)}})$
    $C_{x[i]_2^{(\rho)}} = \mathsf{Com}(x[i]_2^{(\rho)}, r_{x[i]_2^{(\rho)}})$
    $C_{x[i]_3^{(\rho)}} = \mathsf{Com}(x[i]_3^{(\rho)}, r_{x[i]_3^{(\rho)}})$
$(st_x, a_x) \leftarrow \mathcal{P}\{\prod_{i \in [0, |x|-1]} C_{2^i \cdot x[i]} \equiv C_x\}.Commit(x, \sum_{i=0}^{|x|-1} 2^i \cdot r_i, r)$
foreach $i \in [0, |x| - 1]$ **do**
  $(st_{x[i]}, a_{x[i]}) \leftarrow \mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Commit(x[i], r_i)$
  foreach $\rho \in [1, t]$ **do**
    $C_{x[i]^{(\rho)}} = C_{x[i]_1^{(\rho)}} \cdot C_{x[i]_2^{(\rho)}} \cdot C_{x[i]_3^{(\rho)}}$
    $(st_{x[i]^{(\rho)}}, a_{x[i]^{(\rho)}}) \leftarrow \mathcal{P}\{(C_{x[i]^{(\rho)}} \equiv C_{x[i]}) \vee (C_{x[i]^{(\rho)}} \equiv C_{2+x[i]})\}.Commit($
      $x[i]_1^{(\rho)} + x[i]_2^{(\rho)} + x[i]_3^{(\rho)}, r_{x[i]_1^{(\rho)}} + r_{x[i]_2^{(\rho)}} + r_{x[i]_3^{(\rho)}}, r_i)$
// output of (Commit step)
$a = (a_\zeta, (C_{x[i]})_{|x|}, (C_{x[i]_1^{(\rho)}})_{|x| \cdot t}, (C_{x[i]_2^{(\rho)}})_{|x| \cdot t}, (C_{x[i]_3^{(\rho)}})_{|x| \cdot t},$
      $a_x, (a_{x[i]})_{|x|}, (a_{x[i]^{(\rho)}})_{|x| \cdot t})$
// (*Challenge* step)
$e \leftarrow H(a)$
// (*Response* step)
$\mathbf{r}_\zeta \leftarrow \mathsf{ZKB}_F.Response(e, st_\zeta)$
$\mathbf{r}_x \leftarrow \mathcal{P}\{\prod_{i \in [0, |x|-1]} C_{2^i \cdot x[i]} \equiv C_x\}.Response(e, st_x)$
foreach $i \in [0, |x| - 1]$ **do**
  $\mathbf{r}_{x[i]} \leftarrow \mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Response(e, st_{x[i]})$
  foreach $\rho \in [1, t]$ **do**
    $\mathbf{r}_{x[i]^{(\rho)}} \leftarrow \mathcal{P}\{(C_{x[i]^{(\rho)}} \equiv C_{x[i]}) \vee (C_{x[i]^{(\rho)}} \equiv C_{2+x[i]})\}.Response(e, st_{x[i]^{(\rho)}})$
return $(e, a, \mathbf{r}_\zeta, \mathbf{r}_x, (\mathbf{r}_{x[i]})_{|x|}, (\mathbf{r}_{x[i]^{(\rho)}})_{|x| \cdot t},$
      $(x[i]_e^{(\rho)}, r_{x[i]_e^{(\rho)}})_{|x| \cdot t}, (x[i]_{e+1}^{(\rho)}, r_{x[i]_{e+1}^{(\rho)}})_{|x| \cdot t})$

Fig. 1: Description of Cross-ZKB++ (First Attempt) Prove algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++.

$\{Reject, Accept\} \leftarrow \mathsf{Verify}(C_x, p)$

1 : // *Reconstruct* step
2 : Parse p as $(e, a, \mathbf{r}_\zeta, \mathbf{r}_x, (\mathbf{r}_{x[i]})_{|x|}, (\mathbf{r}_{x[i]^{(\rho)}})_{|x|\cdot t},$
3 : $\quad (x[i]_e^{(\rho)}, r_{x[i]_e^{(\rho)}})_{|x|\cdot t}, (x[i]_{e+1}^{(\rho)}, r_{x[i]_{e+1}^{(\rho)}})_{|x|\cdot t})$
4 : Parse a as $(a_\zeta, (C_{x[i]})_{|x|}, (C_{x[i]_1^{(\rho)}})_{|x|\cdot t}, (C_{x[i]_2^{(\rho)}})_{|x|\cdot t}, (C_{x[i]_3^{(\rho)}})_{|x|\cdot t},$
5 : $\quad a_x, (a_{x[i]})_{|x|}, (a_{x[i]^{(\rho)}})_{|x|\cdot t})$
6 : **foreach** $i \in [0, |x| - 1]$ **do**
7 : $\quad$ **foreach** $\rho \in [1, t]$ **do**
8 : $\quad\quad$ *Reject* if $C_{x[i]_e^{(\rho)}} \neq \mathsf{Com}(x[i]_e^{(\rho)}, r_{x[i]_e^{(\rho)}})$ or $C_{x[i]_{e+1}^{(\rho)}} \neq \mathsf{Com}(x[i]_{e+1}^{(\rho)}, r_{x[i]_{e+1}^{(\rho)}})$
9 : $(st'_\zeta, a'_\zeta) \leftarrow \mathsf{ZKB}_F.Reconstruct(e, \mathbf{r}_\zeta)$
10 : *Reject* if $(x_e^{(\rho)})_t, (x_{e+1}^{(\rho)})_t$ do not match respective values in st'_ζ
11 : $a'_x \leftarrow \mathcal{V}\{\prod_{i \in [0, |x|-1]} C_{2^i \cdot x[i]} \equiv C_x\}.Reconstruct(e, \mathbf{r}_x)$
12 : **foreach** $i \in [0, |x| - 1]$ **do**
13 : $\quad a'_{x[i]} \leftarrow \mathcal{V}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Reconstruct(e, \mathbf{r}_{x[i]})$
14 : $\quad$ **foreach** $\rho \in [1, t]$ **do**
15 : $\quad\quad C_{x[i]^{(\rho)}} = C_{x[i]_1^{(\rho)}} \cdot C_{x[i]_2^{(\rho)}} \cdot C_{x[i]_3^{(\rho)}}$
16 : $\quad\quad a'_{x[i]^{(\rho)}} \leftarrow \mathcal{V}\{(C_{x[i]^{(\rho)}} \equiv C_{x[i]}) \vee (C_{x[i]^{(\rho)}} \equiv C_{2+x[i]})\}.Reconstruct(e, \mathbf{r}_{x[i]^{(\rho)}})$
17 : $a' = (a'_\zeta, (C_{x[i]})_{|x|}, (C_{x[i]_1^{(\rho)}})_{|x|\cdot t}, (C_{x[i]_2^{(\rho)}})_{|x|\cdot t}, (C_{x[i]_3^{(\rho)}})_{|x|\cdot t},$
18 : $\quad a'_x, (a'_{x[i]})_{|x|}, (a'_{x[i]^{(\rho)}})_{|x|\cdot t})$
19 : $e' \leftarrow H(a')$
20 : *Accept* if $e' = e$, otherwise *Reject*.

Fig. 2: Description of Cross-ZKB++ (First Attempt) Verify algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++.

1. First it uses the 3-special soundness of the ZKB++ protocol to extract a value x_{ZKB} for which $F(x_{ZKB}) = 1$.
2. It uses the 2-special soundness of the Σ-protocols for the algebraic statements to extract the values $x[i]$, for all $i \in [0, |x|-1]$, and the corresponding opening information $r_{x[i]}$.

We now show the rest of his steps. Without loss of generality, let us assume that $e_1 = 1, e_2 = 2$ and $e_3 = 3$. For all $i \in [0, |x|-1]$ let $w_2 = (x, r, x[i], r_{x[i]}, x[i]_1, r_{x[i]_1}, x[i]_2, r_{x[i]_2}, x[i]_3, r_{x[i]_3})$ be the witness extracted in step 2 and $w_1 = (x_{ZKB})$ be the witness extracted in step 1. Moreover, for $i \in \{1, 2, 3\}$ let $r_{x[i]_{e_i}}$ and $r_{x[i]_{e_i+1}}$ be the opening information to commitments $C_{x[i]_{e_i}}$ and $C_{x[i]_{e_i+1}}$, where we know that $C_{x[i]_{e_i}} = \mathsf{Com}(x[i]_{e_i}, r_{x[i]_{e_i}})$ and $C_{x[i]_{e_i+1}} = \mathsf{Com}(x[i]_{e_i+1}, r_{x[i]_{e_i+1}})$.

We now turn to the following observation. If at some point the algorithm Extr_{Cross} encounters two different opening information to one commitment, i.e. $\mathsf{Com}(a, b) = \mathsf{Com}(c, d)$ it can use (a, b, c, d) to compute the equivocal trapdoor and open any commitment to an arbitrarily value. In particular, it can use this trapdoor to open commitment C_x to the value x_{ZKB}, i.e. in case $x \neq x_{ZKB}$ we can use (x, r) and the equivocal trapdoor to compute $x^* = x_{ZKB}$ and the corresponding r^* such that $C_x = \mathsf{Com}(x^*, r^*)$, which would constitute a valid witness w^*.

We now proceed with the proof and notice that due to the verification done by the verifier and the extracted witness w_2, we know that

$$C_{x[i]_1} = \mathsf{Com}(x[i]_1, r_{x[i]_1}) = \mathsf{Com}(x[i]_{e_1}, r_{x[i]_{e_1}}) = \mathsf{Com}(x[i]_{e_3+1}, r_{x[i]_{e_3+1}}),$$
$$C_{x[i]_2} = \mathsf{Com}(x[i]_2, r_{x[i]_2}) = \mathsf{Com}(x[i]_{e_2}, r_{x[i]_{e_2}}) = \mathsf{Com}(x[i]_{e_1+1}, r_{x[i]_{e_1+1}}),$$
$$C_{x[i]_3} = \mathsf{Com}(x[i]_3, r_{x[i]_3}) = \mathsf{Com}(x[i]_{e_3}, r_{x[i]_{e_3}}) = \mathsf{Com}(x[i]_{e_2+1}, r_{x[i]_{e_2+1}}),$$

and that for $i \in \{1, 2, 3\}$ $x[i]_{e_i}$ are bits that correspond to disclosed views in the ZKB++ protocol. Thus, it follows that $x[i]_1 = x[i]_{e_1}$, $x[i]_2 = x[i]_{e_2}$ and $x[i]_3 = x[i]_{e_3}$ and in particular that $x_{ZKB}[i] = x[i]_1 \oplus x[i]_2 \oplus x[i]_3$ for all $i \in [0, |x| - 1]$.

We will now argue that because of the soundness of the proof system used in step 2, for all $i \in [0, |x| - 1]$ we have $x[i] = x[i]_1 \oplus x[i]_2 \oplus x[i]_3 = x_{ZKB}[i]$. Let us take a look at the following table.

$x[i]_1$	$x[i]_2$	$x[i]_3$	$x[i]_1 + x[i]_2 + x[i]_3$	$x[i]_1 + x[i]_2 + x[i]_3 - 2$	$x[i]_1 \oplus x[i]_2 \oplus x[i]_3$
0	0	0	0	-2	0
0	0	1	1	-1	1
0	1	0	1	-1	1
0	1	1	2	0	0
1	0	0	1	-1	1
1	0	1	2	0	0
1	1	0	2	0	0
1	1	1	3	1	1

The two rows $x[i]_1 + x[i]_2 + x[i]_3$ and $x[i]_1 + x[i]_2 + x[i]_3 - 2$ correspond to the value that the commitment $C_{x[i]} = \mathsf{Com}(x[i], r_{x[i]})$ can be opened to.

However, due to the fact that the statement contains the additional constraint that the commitment opens to a bit, we conclude that for $(x[i], r_{x[i]})$ we have $x[i] = x_{ZKB}[i]$ (we used the coloured background to highlight the only way that witness w_2 can be correct).

Finally, we know that since the witness w_2 is correct, it follows that:

$$\sum_{i \in [0,|x|-1]} 2^i \cdot x[i] = x.$$

However, since $x[i]$ is the i-th bit of x_{ZKB} this means that $x_{ZKB} = x$ and the Extr_{Cross} can return $w^* = (x^*, r^*) = (x_{ZKB}, \sum_{i \in [0,|x|-1]} 2^i \cdot r_{x[i]})$, which is a valid opening for C_x, where $F(x^*) = 1$.

We conclude that the values returned by Extr_{Cross} are a valid witness for Statement 1.

□

Theorem 2. *Assuming the ZKB++ protocol and the Σ-protocols for the algebraic statements are HVZK and the commitment scheme is perfectly hiding, then Construction 1 is a HVZK Σ-protocol for Statement 1.*

Proof. We will show how to construct a simulator $\mathcal{SIM}$ that on input in Statement 1, outputs a transcript (a, e, z). The simulator works as follows:

- It runs the simulator for ZKB++ receiving a transcript (a', e, z'), where z' contains all the bits $x[i]_e$ and $x[i]_{e+1}$. $\mathcal{SIM}$ chooses open information $r_{x[i]_e}$, $r_{x[i]_{e+1}}$ and computes commitments $C_{x[i]_e} = \mathsf{Com}(x[i]_e, r_{x[i]_e})$, $C_{x[i]_{e+1}} = \mathsf{Com}(x[i]_{e+1}, r_{x[i]_{e+1}})$. Note that the openings $r_{x[i]_e}, r_{x[i]_{e+1}}$ are part of the response z. Commitment to the bits $x[i]$ and the bits $x[i]_{e+2}$ are not opened, so the simulator can compute $C_{x[i]}$ and $C_{x[i]_{e+2}}$ as commitments to zero.
- $\mathcal{SIM}$ runs the simulator for the Σ-protocol for the algebraic statements receiving (a'', e'', z''). Note that since this simulator should work for all possible challenges, there is a non-negligible probability that $e'' = e$. Otherwise, $\mathcal{SIM}$ just restarts it.
- Finally, $\mathcal{SIM}$ sets $a = (a', a'', \{C_{x[i]}, C_{x[i]_1}, C_{x[i]_2}, C_{x[i]_3}\}_{i \in [0,|x|-1]})$ and $z = (z', z'', \{r_{x[i]_e}, r_{x[i]_{e+1}}\}_{i \in [0,|x|-1]})$

Since all the simulators used by $\mathcal{SIM}$ generate valid transcripts it remains to show that the commitments $C_{x[i]}$ and $C_{x[i]_{e+2}}$ generated by $\mathcal{SIM}$ are indistinguishable from values in real transcripts. However, this follows directly by the perfectly hiding property of the commitment scheme. □

Lemma 2. *The soundness error of the Σ-protocol presented in Construction 1 is $2/3$ and it has to be executed $\lambda/(\log_2(3)-1)$ times/rounds to achieve a soundness error of $2^{-\lambda}$.*

Proof. The soundness error is implied directly from 3-special soundness of the protocol (Theorem 1) and the challenge space of cardinality 3. The number of rounds, let us denote it t, is simply the solution of equation $(2/3)^t = 2^{-\lambda}$. □

3.2 Improved Version

The main disadvantage of Construction 1 is that we have to compute $O(|x| \cdot t)$ commitments, which influences the number of public key operations we have to additionally compute. The $|x|$ factor is present because for each round $\rho \in [1, t]$ the relation $x[i]_1^{(\rho)} \oplus x[i]_2^{(\rho)} \oplus x[i]_3^{(\rho)} = x[i]^{(\rho)}$ is expressed as a conjunction of two possible statements and we commit to the bits of the input x in every round. In the following, we optimize Construction 1 to increase efficiency by decreasing the number of commitment to $O(|x| + t)$.

Firstly, we notice that we can use the same commitments to bits of x for every round that we repeat the protocol and instead of committing to the bits of the ZKB++ shares we actually compute commitment to the whole values, saving a lot of computations. Note that this idea will only work if the input to ZKB++ is smaller that the order of the algebraic group that we use, otherwise the bitwise exclusive-or of those values will not constitute a accepting input to the ZKB++ circuit (i.e. $x_1 \oplus x_2 \oplus x_3$ is not always equal to $(x_1 \mod q) \oplus (x_2 \mod q) \oplus (x_3 \mod q)$). However, in the next subsection we show how to make the protocol work for ZKB++ input without a size constraint.

Secondly, the bits $x[i]_{e^{(\rho)}}^{(\rho)}$ and $x[i]_{e^{(\rho)}+1}^{(\rho)}$ are revealed in the response step of the ZKB++ protocol (Fig. 5). Based on this observation, we can change the relation

$$x[i]_1^{(\rho)} \oplus x[i]_2^{(\rho)} \oplus x[i]_3^{(\rho)} = x[i]$$

and express the third share using the hidden value x, i.e.

$$x[i]_{e^{(\rho)}+2}^{(\rho)} = x[i] \oplus (x[i]_{e^{(\rho)}}^{(\rho)} \oplus x[i]_{e^{(\rho)}+1}^{(\rho)}).$$

We now take into account that this relation is constructed for known bits and that we can express $C_{a \oplus \alpha}$ for a given C_a and α using homomorphic properties of the commitment scheme. Thus, we can actually compute a commitment to $x[i]_{e^{(\rho)}+2}^{(\rho)}$ using the commitments to bits of x and the revealed bits of values $x[i]_{e^{(\rho)}}^{(\rho)}$ and $x[i]_{e^{(\rho)}+1}^{(\rho)}$. We use this commitment to bind the value $x[i]_1^{(\rho)} \oplus x[i]_2^{(\rho)} \oplus x[i]_3^{(\rho)}$ with the value x inside the commitment C_x.

In Construction 2 we describe those ideas in more detail. We will show a single round of the protocol, which only has a soundness error of $2/3$ but below present the idea how decrease the soundness error efficiently. Our protocol is divided into four essential steps:

1. committing to bits of x,
2. proving using a Schnorr based Σ-protocol that those commitments contain a bit,
3. a ZKB++ proof that there exists a x_{ZKB} such that $F(x_{ZKB}) = 1$, and
4. constant number of commitments $C_{x_1}, C_{x_2}, C_{x_3}$, which ensure $x = x_{ZKB}$.

Thus, if one would run the protocol many times, this still would require the computation of $O(|x| \cdot t)$ commitments.

We solve this problem by taking advantage of the fact that Schnorr based Σ-protocols can use a larger challenge space that decreases the soundness error without repeating the protocol. Unfortunately, this does not apply for the ZKB++ part and for this to work we have to use a special kind of challenge. Let $e_1, \ldots, e_\rho$ be the challenges used for the ρ runs of the ZKB++ protocol, then we can use e.g. $e_\Sigma = \sum_{i \in [0,\rho-1]} 3^i \cdot e_{i+1}$ in step 2. In other words, we execute the first two steps once using the challenge e_Σ and simultaneously run the last two steps ρ-times, where each ZKB++ execution challenged respectively using $e_1, \ldots, e_\rho$.

This simple trick allows us to increase the efficiency of the proof. Now the prover only has to compute a constant number of commitments per round and commit to the bits of the input x only once.

Construction 2 (Cross-ZKB++) *In the following, we describe necessary steps to add to the ZKB++ protocol (Fig. 5) in order to realize the connection between the input bits $x = (\ldots, x[1], x[0])$ to the function F, where $x < q$ and the public commitment C_x in group of order q, as defined in Statement 1.*

- *(Commit Phase) — The prover executes the commit step of the ZKB++ protocol using input x, where $C_x = \mathsf{Com}(x, r)$. The prover chooses random opening informations $r_1, \ldots r_{|x|-1}$ and commits to the bits $x[i]$ by computing:*

$$C_{x[i]} = \mathsf{Com}(x[i], r_i), \text{ for } i \in [1, |x| - 1].$$

  To compute the remaining commitment he uses the opening information $r_0 = r - \sum_{i \in [1,|x|-1]} 2^i \cdot r_i$. Note that because of the homomorphic properties of the commitment scheme this means that $C_x = \prod_{i \in [0,|x|-1]} [2^i] C_{x[i]} = \mathsf{Com}(2^i \cdot x[i], 2^i \cdot r_i)$. For each bit i of input x the prover executes the commit step of a Σ-protocol for the following algebraic statement:

$$\mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}. \tag{2}$$

  The next step is also different. In this protocol we commit to the full values of the respective input shares x_1, x_2, x_3 and get $C_{x_1}, C_{x_2}, C_{x_3}$, where $C_{x_1} = \mathsf{Com}(x_1, r_{x_1})$, $C_{x_2} = \mathsf{Com}(x_2, r_{x_2})$, $C_{x_3} = \mathsf{Com}(x_3, r_{x_3})$. The prover sends commitments $\{C_{x[i]}\}_{i \in [0,|x|-1]}$, $C_{x_1}, C_{x_2}, C_{x_3}$ and the commitments from the ZKB++ protocol and the Σ-protocol Eq. (2) to the verifier.
- *(Challenge Phase) — The verifier sends the challenge $e \in \{1, 2, 3\}$ to the prover.*
- *(Response Phase) — The prover executes the response step for ZKB++, the Σ-protocol and sends the result to the verifier. Knowing e, the prover computes $\alpha = x_e \oplus x_{e+1}$, where by $\alpha[i]$ we will denote its i-th bit. Using the homomorphic exclusive-or described in subsection 2.1, he then computes the commitment*

$$C_z = \prod_{i \in [0,|x|-1]} [2^i] C_{x[i] \oplus \alpha[i]},$$

which is

$$\mathsf{Com}(x_{e+2}, \sum_{i\in[0,|x|-1]} (-1)^{\alpha[i]} r_{x[i]}).$$

Finally, the prover sends the opening information r_{x_e}, $r_{x_{e+1}}$ for commitments C_{x_e}, $C_{x_{e+1}}$ and value $r_z = r_{x_{e+2}} - \prod_{i\in[0,|x|-1]}(-1)^{\alpha[i]} r_{x[i]}$.

To verify the result the verifier follows the steps specified by the ZKB++ protocol and additionally performs the following checks: reject if the opening is wrong or the shares in the commitments do not match the ones in the ZKB++ views, or if any of the additional algebraic proofs is invalid. The verifier aborts if $C_x \neq \prod_{i\in[0,|x|-1]}[2^i]C_{x[i]}$. Knowing the shares x_e, x_{e+1} and the openings r_{x_e}, $r_{x_{e+1}}$, the verifier also computes $C_z = \prod_{i\in[0,|x|-1]}[2^i]C_{x[i]\oplus\alpha_i}$ and checks that $C_z \cdot \mathsf{Com}(0, r_z) = C_{x_{e+2}}$.

We present in Figs. 3 and 4 the detailed description of Construction 2, instantiated with t rounds of ZKB++ and made non-interactive using the Fiat-Shamir transformation.

Security analysis

Lemma 3. *Assuming the ZKB++ protocol is complete, the Σ-protocols for the algebraic statements are complete and the used commitment scheme is homomorphic, then Construction 2 is a complete Σ-protocol for the statement in Problem 1.*

Proof. Follows by inspection. □

Theorem 3. *Assuming the ZKB++ protocol is 3-special sound the used Σ-protocols are 2-special sound and the used commitment scheme is homomorphic and equivocal, then Construction 2 is a 3-special sound Σ-protocol for Statement 1.*

Proof. As in the proof of Theorem 1 we will construct an efficient algorithm Extr_{Cross} that using 3 accepting tuples (a, e_1, z_1), (a, e_2, z_2) and (a, e_3, z_3) can compute a witness that the statement is true. The extraction algorithm will return a value x and an opening information r such that $F(x) = 1$ and $C_x = \mathsf{Com}(x, r)$, which is a valid witness for the proven statement. We will now describe the idea behind the algorithm Extr_{Cross}, which is as follows:

- First it uses the 3-special soundness of the ZKB++ protocol to extract a value x_{ZKB} for which $F(x_{ZKB}) = 1$.
- It uses the 2-special soundness of the proof system for Eq. (2) to extract the bits $x[i]$, for all $i \in [0, |x| - 1]$, and the opening information $r_{x[i]}$.
- It computes r_{ZKB}, as described below, and returns $(x^*, r^*) = (x_{ZKB}, r_{ZKB})$ as a valid witness.

$p \leftarrow \mathsf{Prove}(x, C_x = \mathsf{Com}(x, r))$

1 : // (*Commit* step)
2 : $(st_\zeta, a_\zeta) \leftarrow \mathsf{ZKB}_F.Commit(x)$
3 : **foreach** $i \in [1, |x| - 1]$ **do**
4 : $\quad C_{x[i]} = \mathsf{Com}(x[i], r_i)$
5 : $r_0 = r - \sum_{i \in [1,,|x|-1]} 2^i \cdot r_i$
6 : $C_{x[0]} = \mathsf{Com}(x[0], r_0)$
7 : **foreach** $i \in [0, |x| - 1]$ **do**
8 : $\quad (st_{x[i]}, a_{x[i]}) \leftarrow \mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Commit(x[i], r_i)$
9 : **foreach** $\rho \in [1, t]$ **do**
10 : $\quad$ Extract shares $x_1^{(\rho)}, x_2^{(\rho)}, x_3^{(\rho)}$ from st_ζ
11 : $\quad C_{x_1^{(\rho)}} = \mathsf{Com}(x_1^{(\rho)}, r_{x_1^{(\rho)}}), C_{x_2^{(\rho)}} = \mathsf{Com}(x_2^{(\rho)}, r_{x_2^{(\rho)}}), C_{x_3^{(\rho)}} = \mathsf{Com}(x_3^{(\rho)}, r_{x_3^{(\rho)}})$
12 : $a = (a_\zeta, (C_{x[i]})_{|x|}, (a_{x[i]})_{|x|}, (C_{x_1^{(\rho)}})_t, (C_{x_2^{(\rho)}})_t, (C_{x_3^{(\rho)}})_t)$ // output of (Commit step)
13 : // (*Challenge* step)
14 : $e \leftarrow H(a)$
15 : // (*Response* step)
16 : $\mathbf{r}_\zeta \leftarrow \mathsf{ZKB}_F.Response(e, st_\zeta)$
17 : **foreach** $i \in [0, |x| - 1]$ **do**
18 : $\quad \mathbf{r}_{x[i]} \leftarrow \mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Response(e, st_{x[i]})$
19 : **foreach** $\rho \in [1, t]$ **do**
20 : $\quad \alpha^{(\rho)} = x_e^{(\rho)} \oplus x_{e+1}^{(\rho)}$
21 : $\quad C_z^{(\rho)} = \prod_{i \in [0,|x|-1]} [2^i] C_{x[i] \oplus \alpha^{(\rho)}[i]}$
22 : $\quad r_z^{(\rho)} = r_{x_{e+2}^{(\rho)}} - \prod_{i \in [0,|x|-1]} (-1)^{\alpha[i]} r_{x[i]}$
23 : **return** $(e, a, \mathbf{r}_\zeta, (\mathbf{r}_{x[i]})_{|x|}, (x_e^{(\rho)}, r_{x_e^{(\rho)}})_t, (x_{e+1}^{(\rho)}, r_{x_{e+1}^{(\rho)}})_t, (r_z^{(\rho)})_t)$

Fig. 3: Description of Cross-ZKB++ Prove algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++.

$\underline{\{Reject, Accept\} \leftarrow \mathsf{Verify}(C_x, p)}$

1 : // *Reconstruct* step

2 : Parse p as $(e, a, \mathbf{r}_\zeta, (\mathbf{r}_{x[i]})_{|x|}, (x_e^{(\rho)}, r_{x_e^{(\rho)}})_t, (x_{e+1}^{(\rho)}, r_{x_{e+1}^{(\rho)}})_t, (r_z^{(\rho)})_t)$

3 : Parse a as $(a_\zeta, (C_{x[i]})_{|x|}, (a_{x[i]})_{|x|}, (C_{x_1^{(\rho)}})_t, (C_{x_2^{(\rho)}})_t, (C_{x_3^{(\rho)}})_t)$

4 : *Reject* if $C_x \neq \prod_{i \in [0,|x|-1]} [2^i] C_{x[i]}$

5 : **foreach** $\rho \in [1, t]$ **do**

6 : *Reject* if $C_{x_e^{(\rho)}} \neq \mathsf{Com}(x_e^{(\rho)}, r_{x_e^{(\rho)}})$ or $C_{x_{e+1}^{(\rho)}} \neq \mathsf{Com}(x_{e+1}^{(\rho)}, r_{x_{e+1}^{(\rho)}})$

7 : $\alpha^{(\rho)} = x_e^{(\rho)} \oplus x_{e+1}^{(\rho)}$

8 : $C_z^{(\rho)} = \prod_{i \in [0,|x|-1]} [2^i] C_{x[i] \oplus \alpha^{(\rho)}[i]}$

9 : *Reject* if $C_z^{(\rho)} \cdot \mathsf{Com}(0, r_z^{(\rho)}) \neq C_{x_{e+2}^{(\rho)}}$

10 : $(st'_\zeta, a'_\zeta) \leftarrow \mathsf{ZKB}_F.Reconstruct(e, \mathbf{r}_\zeta)$

11 : *Reject* if $(x_e^{(\rho)})_t, (x_{e+1}^{(\rho)})_t$ do not match respective values in st'_ζ

12 : **foreach** $i \in [0, |x| - 1]$ **do**

13 : $a'_{x[i]} \leftarrow \mathcal{V}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Reconstruct(e, \mathbf{r}_{x[i]})$

14 : $a' = (a'_\zeta, (C_{x[i]})_{|x|}, (a'_{x[i]})_{|x|}, (C_{x_1^{(\rho)}})_t, (C_{x_2^{(\rho)}})_t, (C_{x_3^{(\rho)}})_t)$

15 : $e' \leftarrow H(a')$

16 : *Accept* if $e' = e$, otherwise *Reject*.

Fig. 4: Description of Cross-ZKB++ Verify algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++.

We will now show how Extr_{Cross} computes witness $w^* = (x^*, r^*)$ and that the returned values are valid. Let $w_2 = (\{x[i], r_{x[i]}\}_{i\in[0,|x|-1]})$ be the witness extracted in step 2 and $w_1 = (x_{ZKB})$ be the witness extracted in step 1. Moreover, let r_{x_e} and $r_{x_{e+1}}$ be the opening information to commitments $C_{x_{e_1}}$, $C_{x_{e_2}}$ and $C_{x_{e_3}}$, where we know that $C_{x_{e_1}} = \mathsf{Com}(x_{e_1}, r_{x_{e_1}})$, $C_{x_{e_2}} = \mathsf{Com}(x_{e_2}, r_{x_{e_2}})$ and $C_{x_{e_3}} = \mathsf{Com}(x_{e_3}, r_{x_{e_3}})$.

Again we observe that if the algorithm Extr_{Cross} encounters two different opening information to one commitment the equivocal trapdoor can be used to open the commitment C_x to the value $w_1 = x_{ZKB}$.

We now proceed with the proof and notice that since all the tuples are accepting, we conclude that the openings of the commitments $C_{x_1}, C_{x_2}, C_{x_3}$ are valid. This is the case because we have valid openings $(x_{e_1}, r_{x_{e_1}})$, $(x_{e_2}, r_{x_{e_2}})$, $(x_{e_3}, r_{x_{e_3}})$. It follows that the binary representations of x_1, x_2, x_3 correspond to the correct input of the ZKB++ protocol and we have $x_{ZKB} = x_1 \oplus x_2 \oplus x_3$. Note that this is only true because x_{ZKB} is shorter that the order of the used group. Moreover, we know that by construction:

$$C_x = \prod_{i\in[0,|x|-1]} [2^i]\mathsf{Com}(x[i], r_{x[i]}),$$

and that $x[i]$ are bits.

Let $e = e_1$, the Extr_{Cross} computes commitment $C_z = \prod_{i\in[0,|x|-1]}[2^i]C_{x[i]\oplus\alpha_i}$, where $\alpha = x_e \oplus x_{e+1}$. Since we know that for e_1 we receive an accepting state, we know that $C_z \cdot \mathsf{Com}(0, r_z) = C_{x_{e+2}} = \mathsf{Com}(x_{e+2}, r_{x_{e+2}})$. This basically means that Extr_{Cross} can open C_z to x_{e+2} using randomness $r_{x_{e+2}} - r_z$. We now distinguish two cases:

1. the openings of C_z and $C_{x_{e+2}}$ are different, i.e. this means that
$$\sum_{i\in[0,|x|-1]} 2^i(x[i] \oplus \alpha_i) \neq x_{e+2},$$
2. the openings of C_z and $C_{x_{e+2}}$ are the same.

In the first case we notice that Extr_{Cross} knows openings of the commitment C_z to two different values. Thus, it can use an extractor $\mathsf{Extr}_{\mathsf{ck}}$ to compute the equivocality trapdoor for the commitment scheme and compute r_{ZKB} as $\mathsf{Eval}(\tau, x_{ZKB}, (C_x, \sum_{i\in[0,|x|-1]} 2^i x[i], \sum_{i\in[0,|x|-1]} 2^i r_{x[i]}))$. In other words, the extraction algorithm Extr_{Cross} uses the trapdoor to open the commitment from the statement to the value x_{ZKB} for which $F(x_{ZKB}) = 1$. This means that the returned values are a valid witness for the proven statement. In the second case we know that:

$$\sum_{i\in[0,|x|-1]} 2^i(x[i] \oplus \alpha_i) = x_{e+2}.$$

This means that $r^* = \sum_{i\in[0,|x|-1]} 2^i r_{x[i]}$ is an opening of the commitment C_x to a value x' for which we know that $x' \oplus x_e \oplus x_{e+1} = x_{e+2}$. It follows that $x' = x_e \oplus x_{e+1} \oplus x_{e+2} = x_{ZKB}$. Thus, in this case Extr_{Cross} can also return $w^* = (x_{ZKB}, r^*)$, which ends the proof. □

Theorem 4. *Assuming the ZKB++ protocol is HVZK and the commitment scheme is perfectly hiding, then Construction 2 is a HVZK Σ-protocol for Statement 1.*

Proof. We will show how to construct a simulator $\mathcal{SIM}$ that on input of a statement as in Statement 1 with commitment C_x, outputs a transcript (a, e, z). The simulator works as follows:

- It runs the simulator for ZKB++ receiving a transcript (a', e, z'), where z' contains the shares x_e and x_{e+1}.
- $\mathcal{SIM}$ chooses randomness r_{x_e}, $r_{x_{e+1}}$ and computes commitments $C_{x_e} = \mathsf{Com}(x_e, r_{x_e})$, $C_{x[i]_{e+1}} = \mathsf{Com}(x_{e+1}, r_{x_{e+1}})$. Note that the openings for those commitments are part of the response z. Commitments to the bits of $x[i]$ and to x_{e+2} are not opened, so the simulator can compute the commitments $C_{x[i]}$ and $C_{x_{e+2}}$ as follows.
- For $i \in [1, |x| - 1]$ it computes commitments $C_{x[i]}$ as commitments to 0. For $j = 0$ it uses the homomorphic properties of the commitment scheme to compute $C_{x[j]}$ such that $C_x = \prod_{i=0}^{|x|-1} [2^i] C_{x[i]}$.
- It then chooses a randomness r_z and computes

$$C_{x_{e+2}} = \prod_{i \in [0,|x|-1]} C_{2^i \cdot (x[i] \oplus x_e[i] \oplus x_{e+1}[i])} \cdot \mathsf{Com}(0, -(r_z)).$$

- $\mathcal{SIM}$ runs the simulator for the Σ-protocol for the algebraic statement receiving (a'', e'', z''). Note that since this simulator should work for all possible challenges, there is a non-negligible probability that $e'' = e$. Otherwise, $\mathcal{SIM}$ just restarts it.
- Finally, $\mathcal{SIM}$ sets $a = (a', a'', C_{x_1}, C_{x_2}, C_{x_3}, \{C_{x[i]}\}_{i \in [0,|x|-1]})$ and $z = (z', z'', r_{x_e}, r_{x_{e+1}}, r_z)$

□

Lemma 4. *The soundness error of the Σ-protocol presented in Construction 2 is* 2/3.

Proof. It is implied directly from 3-special soundness of the protocol (Theorem 3) and the challenge space of cardinality 3. □

3.3 Optimization for large input space

We now show how to reduce the number of public key operations to be proportional to the message space of the commitment scheme and independent of the input size of the function F, which is desirable when the input to the ZKB++ circuit is large and required if we want to use Construction 2 for such circuits. This optimization will utilize the properties of modular arithmetics.

Let $F(m) = 1$ be a function that has to be proven in the cross-domains, and let $m \geq q$ where $[0, q-1]$ is the message space of the commitment scheme. The prover proceeds as follows. Instead of committing to m, it commits to $C =$

$\mathsf{Com}_q(m')$, where m' satisfies $m' < q$ and $m = k \cdot q + m'$ and proves the relation between m and m' as part of F. Let the original cross-domain statement be described as: $\mathcal{P}\{m : (F(m) = 1) \wedge (C_m = \mathsf{Com}_q(m, r))\}$. Then the optimized version is defined as:

$$\mathcal{P}\{m, m', k : (F_{opt}(m, m', k, q) = 1) \wedge C_m = \mathsf{Com}_q(m', r)\},$$

where

$$F_{opt}(m, m', k, q) = (F(m) = 1 \wedge (m = m' + k \cdot q)) .$$

It is easy to see that C_m can be opened either to m, or to m', as both values are equal modulo q. Furthermore, the prover indeed proves that m and m' are equal modulo q. Finally, the prover proves that m' is the value committed to in C_m.

This solution requires us to create an arithmetic circuit as part of the statement proven by ZKB++. Fortunately, this is a standard integer multiplication circuit of a number $k < |x|$ and $q = O(\lambda)$. We can view such a multiplication as the addition of q, k-bit numbers. Since adding two k-bit numbers can be done using $O(k)$ gates, it follows that this multiplication can be done using $O(k \cdot q)$ gates, which is also $O(|x| \cdot \lambda)$. In particular, we have that this can be done using $O(|F| \cdot \lambda)$ gates, because $|x| < |F|$. Thus, the asymptotic number of symmetric operations remains the same and we only introduce a slight overhead using this technique.

3.4 Efficiency

We will discuss the computation overhead and increase in the proof size of our techniques. We will compare both constructions for Statement 1 and focus only on public key operations, i.e. exponentiations and multiplications in the used group $\mathbb{G}$ of order q, where $\ell q = \log q$. Let us assume that we run both protocols ρ times for input x and that we use Pedersen commitments. Moreover, we will by ℓ_{ZKB} denote the proof size of the ZKB++ protocol, by ℓ_Σ the proof size of the Σ-protocol for Eq. (1) and by $\ell_{\mathbb{G}}$ the size of group elements.

In such a case the proof size of Construction 1 is $\rho \cdot (\ell_{ZKB} + \ell_\Sigma + 4 \cdot |x| \cdot \ell_{\mathbb{G}} + 2 \cdot |x| \cdot \ell q)$, which asymptotically is $O(|x| \cdot \rho)$. Construction 2 was introduced to decrease this by depending less on Σ-protocols for algebraic statements and using the homomorphic properties of the commitment scheme. When executed in parallel, the proof size is $\rho \cdot (\ell_{ZKB} + (3 \cdot |x| + 3) \cdot \ell_{\mathbb{G}} + 2 \cdot |x| \cdot \ell q + 3 \cdot \ell q)$, which is better but still $O(|x| \cdot \rho)$. Fortunately, we have shown that certain parts of the computations can be reused throughout every round. Therefore, for an optimized version of Construction 2 we end up with a proof size of $\rho \cdot (\ell_{ZKB} + 3 \cdot \ell_{\mathbb{G}} + 3 \cdot \ell q) + |x| \cdot (3 \cdot \ell_{\mathbb{G}} + 2 \cdot \ell q)$, which is $O(|x| + \rho)$.

To compute the proof in Construction 1 we have to compute $4 \cdot \rho \cdot |x|$ commitments and compute the proof for statement Eq. (1), which strongly depends on the instantiation but it requires at least $O(\rho \cdot |x|)$ exponentiations. Computing commitments to bits costs one exponentiation and one multiplication. In the end, for this construction we require $O(\rho \cdot |x|)$ exponentiations. In case of Construction 2 we have to compute $|x| \cdot (3 \cdot \ell_{\mathbb{G}}) + \rho \cdot 3 \cdot \ell_{\mathbb{G}}$ commitments and $2 \cdot |x|$

exponentiations for the proof for statement Eq. (2). We also have to compute the commitment C_z, which requires us to compute $|x| \cdot \rho$ multiplications in $\mathbb{G}$. Given the fact, that we assumed that $|x|$ is of the size of $\log q$ it follows that the cost of those multiplications is comparable with ρ exponentiations in $\mathbb{G}$. It follows, that for this construction we require only $O(|x| + \rho)$ exponentiations.

4 NIZK OR-proofs in cross-domains

Proofs of partial knowledge [19], also known as OR-proofs, allow to efficiently prove only a part of a statement, without revealing, which part has been proven. Below we show how to prove the most simple OR-statement in cross-domains, which can be used as a basis for proving more complex statements.

Statement 2 *Prove knowledge of x_1 s.t. $F(x_1) = 1$ or knowledge of x_2 such that $y = g^{x_2}$, where F is an arithmetic circuit.*

We are going to use ZKB++ for proving the first part and the standard Schnorr proof for the second part. Since the both parts of the proof system are Σ-protocols, a challenge e will be "distributed" between these parts as $e = e_1 + e_2$. Assume $e_1 \in \mathbb{Z}_p$ and $e_2 \in \mathbb{Z}_q$, where $p > q$. The prover generates e_1 or e_2 and derives the remaining element based on e. Both e_1 and e_2 should have the same distribution regardless of the part that is being proved. Depending on which part is being proved, we proceed as follows. Given $e \in \mathbb{Z}_p$, to prove the first part the prover picks $e_2 \leftarrow_R \mathbb{Z}_q$ and computes $e_1 = e - k \cdot e_2$, where $k = \lfloor p/q \rfloor$. Given $e \in \mathbb{Z}_p$, to prove the second part the prover picks $e_1 \leftarrow_R \mathbb{Z}_p$ and computes $e_2' = e - e_1 \in \mathbb{Z}_p$. To preserve the distribution of e_2, the prover performs rejection sampling: it further computes the largest p' that satisfies $p' = k \cdot q \leq p$ for integer k and rejects and regenerates e_1 if $e_1 > \mathbb{Z}_{p'}$, otherwise $e_2 \leftarrow e_2' (\text{mod } q)$. It is easy to see that the probability of rejection is at most $1/2$, and e_1 and e_2 are distributed identically regardless of which part has been proven.

Remark 1. If $p >> q$, it suffices to stay in $\mathbb{Z}_p$ and convert an element from $\mathbb{Z}_p$ to $\mathbb{Z}_q$ by taking its residue.

5 Conclusion

Zero-knowledge proofs are an essential component in various protocols, including payment, electronic voting, anonymous credential systems. Proofs based on algebraic groups and for arithmetic circuits represent two different domains. In this work, we presented an efficient Σ-protocol in cross-domains, which can be used to prove the possession of standard RSA/DSA signatures. Moreover, the protocol can be executed non-interactively using the Fiat-Shamir transformation. It follows, that our results can be applied to build round-optimal and concurrent-secure anonymous credentials based on standard signature schemes. Our techniques are especially beneficial when applied for large circuits and when

the prover's running time is critical. As future work, it would be interesting to explore whether the approach by Ames et al. [4] can be used to achieve yet more efficient and compact NIZK proofs in cross-domains.

Acknolwedgements. We would like to thank the anonymous reviewers for their valuable comments. This work was supported by the German Research Foundation (DFG) through funding for the project Methoden und Instrumente zum Verständnis und zur Kontrolle von Datenschutz (SFB1223/1) and by the German Federal Ministry of Education and Research (BMBF) through funding for CISPA and the CISPA-Stanford Center for Cybersecurity (FKZ: 16KIS0762).

References

1. Technical background of version 1 bitcoin addresses. `https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses`, accessed: 2018-10-09
2. Zcash parameter generation. https://z.cash/technology/paramgen.html, accessed: 2018-10-08
3. Agrawal, S., Ganesh, C., Mohassel, P.: Non-interactive zero-knowledge proofs for composite statements. In: Advances in Cryptology–CRYPTO 2018 (3). pp. 643–673. Springer (2018)
4. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 2087–2104. ACM (2017)
5. Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 1087–1098. ACM (2013)
6. Bangerter, E., Camenisch, J., Maurer, U.: Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order. In: Public Key Cryptography–PKC 2005. pp. 154–171. Springer (2005)
7. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 263–280. Springer (2012)
8. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: P-signatures and noninteractive anonymous credentials. In: Theory of Cryptography Conference. pp. 356–374. Springer (2008)
9. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Advances in Cryptology–CRYPTO 2013, pp. 90–108. Springer (2013)
10. Brands, S.A.: Rethinking public key infrastructures and digital certificates: building in privacy. Mit Press (2000)
11. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE Symposium on Security and Privacy (SP). pp. 319–338. IEEE (2018)
12. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 93–118. Springer (2001)

13. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Advances in Cryptology–CRYPTO 2004. pp. 56–72. Springer (2004)
14. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: Advances in Cryptology–CRYPTO '97. pp. 410–424. Springer (1997)
15. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Tech. rep., ETH Zurich, Institut für Theoretische Informatik (1997)
16. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1825–1842. ACM (2017)
17. Chase, M., Ganesh, C., Mohassel, P.: Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In: Advances in Cryptology–CRYPTO 2016. pp. 499–530. Springer (2016)
18. Chaum, D., Evertse, J.H., van de Graaf, J., Peralta, R.: Demonstrating possession of a discrete logarithm without revealing it. In: Advances in Cryptology–CRYPTO '86. pp. 200–212. Springer (1986)
19. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Advances in Cryptology–CRYPTO '94. pp. 174–187. Springer (1994)
20. Escala, A., Groth, J.: Fine-tuning Groth-Sahai proofs. In: Public-Key Cryptography–PKC 2014. pp. 630–649. Springer (2014)
21. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology–CRYPTO '86. pp. 186–194. Springer (1987)
22. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Advances in Cryptology–EUROCRYPT 2013. pp. 626–645. Springer (2013)
23. Giacomelli, I., Madsen, J., Orlandi, C.: Zkboo: Faster zero-knowledge for boolean circuits. In: USENIX Security Symposium. pp. 1069–1083 (2016)
24. Goldreich, O., Micali, S., Wigderson, A.: How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. In: Advances in Cryptology–CRYPTO '86. pp. 171–185. Springer (1986)
25. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on computing **18**(1), 186–208 (1989)
26. Groth, J.: Non-interactive zero-knowledge arguments for voting. In: International Conference on Applied Cryptography and Network Security. pp. 467–482. Springer (2005)
27. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Advances in Cryptology–ASIACRYPT 2010. pp. 321–340. Springer (2010)
28. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Advances in Cryptology–EUROCRYPT 2008, pp. 415–432. Springer (2008)
29. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. pp. 21–30. ACM (2007)
30. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 955–966. ACM (2013)

31. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free xor gates and applications. In: International Colloquium on Automata, Languages, and Programming. pp. 486–498. Springer (2008)
32. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM conference on Electronic commerce. pp. 129–139. ACM (1999)
33. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Advances in Cryptology–CRYPTO '91. pp. 129–140. Springer (1991)
34. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptology **4**(3), 161–174 (1991)
35. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE (1994)
36. Songhori, E.M., Hussain, S.U., Sadeghi, A.R., Schneider, T., Koushanfar, F.: Tinygarble: Highly compressed and scalable sequential garbled circuits. In: 2015 IEEE Symposium on Security and Privacy (SP). pp. 411–428. IEEE (2015)
37. Yao, A.C.C.: How to generate and exchange secrets. In: Proceedings of the 27th Annual Symposium on Foundations of Computer Science. pp. 162–167. IEEE (1986)
38. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Advances in Cryptology–EUROCRYPT 2015. pp. 220–250. Springer (2015)

The prover knows x to a public function F, such that $y = F(x)$, where y is public. t denotes the number of (parallel) rounds.

$\underline{p \leftarrow Prove(x)}$

1. (Commit step) For each round $\rho \in [1, t]$: Sample random tapes $k_1^{(\rho)}$, $k_2^{(\rho)}$, $k_3^{(\rho)}$ and simulate the MPC protocol to get an output view $\mathsf{View}_j^{(\rho)}$ and output share $y_j^{(\rho)}$.

$$
\begin{aligned}
&(x_1^{(\rho)}, x_2^{(\rho)}, x_3^{(\rho)}) \leftarrow \mathsf{Share}(x, k_1^{(\rho)}, k_2^{(\rho)}, k_3^{(\rho)}) \\
&= (G(k_1^{(\rho)}), G(k_2^{(\rho)}), x \oplus G(k_1^{(\rho)}) \oplus G(k_2^{(\rho)})) \\
&\mathsf{View}_j^{(\rho)} \leftarrow \mathsf{Upd}(...\mathsf{Upd}(x_j^{(\rho)}, x_{j+1}^{(\rho)}, k_j^{(\rho)}, k_{j+1}^{(\rho)})...) \\
&y_j^{(\rho)} \leftarrow \mathsf{Output}(\mathsf{View}_j^{(\rho)})
\end{aligned}
$$

   Commit $D_j^{(\rho)} \leftarrow H'(k_j^{(\rho)}, \mathsf{View}_j^{(\rho)})$, let $a^{(\rho)} = (y_1^{(\rho)}, y_2^{(\rho)}, y_3^{(\rho)}, D_1^{(\rho)}, D_2^{(\rho)}, D_3^{(\rho)})$ and let $a = a^{(1)}, \ldots, a^{(t)}$ be the output of this step.
2. Compute the challenge: $e \leftarrow H(a)$. Interpret e such that for $\rho \in [1, t]$, $e^{(\rho)} \in \{1, 2, 3\}$.
3. (Response step) For each round $\rho \in [1, t]$: let $b^{(\rho)} = (y_{e^{(\rho)}+2}^{(\rho)}, D_{e^{(\rho)}+2}^{(\rho)})$ and set $z^{(\rho)} \leftarrow (\mathsf{View}_{e^{(\rho)}+1}^{(\rho)}, k_{e^{(\rho)}}^{(\rho)}, k_{e^{(\rho)}+1}^{(\rho)})$. If $e^{(\rho)} \neq 1$, add $x_3^{(\rho)}$ to $z^{(\rho)}$. Let $\mathbf{r} \leftarrow [(b^{(1)}, z^{(1)}), \ldots, (b^{(t)}, z^{(t)})]$ be the output of this step.
4. Output $p \leftarrow [e, \mathbf{r}]$.

$\underline{b \leftarrow Verify(y, p)}$:

1. (Reconstruct step) For each round $\rho \in [1, t]$: Run the MPC protocol to reconstruct the views. In particular: compute $x_{e^{(\rho)}}^{(\rho)}, x_{e^{(\rho)}+1}^{(\rho)}$ using $z^{(\rho)}$ as part of $\mathbf{r}$ of p in one of the following ways: $x_1^{(\rho)} \leftarrow G(k_1^{(\rho)})$, $x_2^{(\rho)} \leftarrow G(k_2^{(\rho)})$, or $x_3^{(\rho)}$ given as part of $z_{(\rho)}$.
   Obtain $\mathsf{View}_{e^{(\rho)}+1}^{(\rho)}$ from $z_{(\rho)}$ and compute
   $\mathsf{View}_e^{(\rho)} \leftarrow \mathsf{Upd}(...\mathsf{Upd}(x_j^{(\rho)}, x_{j+1}^{(\rho)}, k_j^{(\rho)}, k_{j+1}^{(\rho)})...)$, $y_{e^{(\rho)}}^{(\rho)} \leftarrow \mathsf{Output}(\mathsf{View}_{e^{(\rho)}}^{(\rho)})$, $y_{e^{(\rho)}+1}^{(\rho)} \leftarrow \mathsf{Output}(\mathsf{View}_{e^{(\rho)}+1}^{(i)})$, $y_{e^{(\rho)}+2}^{(i)} \leftarrow y \oplus y_{e^{(\rho)}}^{(i)} \oplus y_{e^{(\rho)}+1}^{(i)}$.
   Compute the commitments for views $\mathsf{View}_{e^{(\rho)}}^{(\rho)}$ and $\mathsf{View}_{e^{(\rho)}}^{(\rho+1)}$. For $j \in \{e^{(\rho)}, e^{(\rho)} + 1\}$:
   $D_j^{(\rho)} \leftarrow H'(k_j^{(\rho)}, \mathsf{View}_j^{(\rho)})$.
   Let $a'^{(\rho)} = (y_1^{(\rho)}, y_2^{(\rho)}, y_3^{(\rho)}, D_1^{(\rho)}, D_2^{(\rho)}, D_3^{(\rho)})$ and note that $y_{e^{(\rho)}+2}^{(\rho)}$ and $D_{e^{(\rho)}+2}^{(\rho)}$ are part of $z_{(\rho)}$. Let $a' = (a'^{(1)}, \ldots, a'^{(t)})$ be the output of this step.
2. Compute the challenge: $e' \leftarrow H(a')$. If $e' = e$, output Accept, otherwise output Reject.

Fig. 5: Non-interactive ZKB++ [16].