ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

UNIVERSITÀ DEGLI STUDI DI UDINE
hic sunt futura

MASTER THESIS IN
ARTIFICIAL INTELLIGENCE AND CYBERSECURITY

Zero-Knowledge friendly cryptographic permutation: theory and implementation

CANDIDATE
Stefano Trevisani

SUPERVISORS
Prof. Alberto Policriti
Dr. Arnab Roy

CO-SUPERVISOR
Prof. Elisabeth Oswald

TUTOR
Msc. Matthias Steiner

Academic Year 2021-2022

Institute Contacts
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

Institute Contacts
Alpen-Adria-Universität Klagenfurt,
Universitätsstraße, 65–67
9020 Klagenfurt am Wörthersee — Austria
+43 463 2700
https://www.aau.at/

Abstract

Zero Knowledge (ZK) proof systems have been an increasingly studied subject in the last 40 years. In the last decade, the efficiency of the proposed frameworks, along with the processing power of computing devices, has improved to the point of making ZK computation feasible in real-world scenarios. One of the primary applications lies in hash-tree commitment verification, and in this past five years there has been intense research in proposing ZK-friendly cryptographic primitives. In this work, we begin by studying the history of ZK systems and reviewing the state of the art concerning ZK-friendly cryptographic permutations. We then present a novel, generic algebraic framework to design cryptographic permutations and we apply it to construct a new permutation. Finally, we implement our permutation together with the reviewed ones in the Groth16 ZK-SNARK framework and compare their efficiency for Merkle-Tree commitment verification.

Contents

1 Introduction

An important research branch of cryptography which emerged in the last fourty years is the study of *Zero Knowledge Interactive* (ZK-I) protocols, and more specifically zero knowledge proof systems (ZKP) [4]. The main idea behind ZKP systems is to have two (or, in some cases, more) parties, where one is the *prover* and the other is the *verifier*: in a classical proof system, the prover must be able to convince the verifier that a certain statement is true, when this is indeed the case, but the verifier cannot be fooled if the statement is actually false. In a ZKP system we also require that the verifier does not get any useful additional information (i.e. knowledge) other than the truth, or lack thereof, of the statement. This additional requirement is particularly interesting when dealing with statements that are notoriously (believed to be) hard to prove, so that the verifier would not be realistically able to prove them in a reasonable amount of time. As a simple example, a prover would like to show that a propositional logic formula is satisfiable (an instance of the famous SAT problem) without revealing the satisfying assignment to the verifier.

Along the years, additional interesting and useful properties have been added to extend and improve the capabilities of ZKP systems. For example, we would like to have a *Non-interactive* (ZK-NP) protocol, to minimize the amount of required communication and have it happen only at the beginning and at the end of the protocol. We could also want to relax the soundness requirement so that it is guaranteed only against computationally bounded provers: in this case, instead of ‘proof’ we use the term *ARgument of Knowledge*, and hence we can have ZK-IARK/ZK-NARK systems. More recently, there has been a research effort towards reducing the length of the ARK by ensuring that it is constant size or at most bounded by a logarithmic function in the length of the theorem statement: such systems are said to be *Succint*. Implementations of ZK-SNARK system, like Pinocchio [9] or Groth16 [7], represent the current state of the art (SoTA) of ZKP systems, and allow to generate proofs to verify any computation representable by means of *bounded arithmetic circuits*. A major downside of ZK-SNARK protocols is their need of a trusted third party (TTP) to setup the system, hence current research is studying *Transparent* systems (ZK-STARK) to address this issue [2].

An especially useful application of ZKP systems is proving knowledge of a preimage for a cryptographic hash function digest (a.k.a. commitment). Many data integrity systems, such as blockchains, rely on Merkle Trees [8] to ensure efficient commitment validation, especially in dynamic environments. In Merkle Trees, an hash function is applied in a bottom-up fashion: the leaves will contain the data owned by some parties, while the root will contain the tree commitment. In a non-ZK setting, a prover would send the verifier his leaf together with the co-path, the verifier would then recompute the tree commitment and compare it with the public one and be convinced whether or not the prover does actually own the leaf. On the other hand, in a ZK-SNARK setting, we first have to represent the computation through a bounded arithmetic circuit, i.e. we are allowed to use exclusively a constant number of additions and multiplications over some suitable finite field. The circuit, together with a *proving key* provided by a TTP, and some private and public data, is then used by the prover to generate a proof which is sent to the verifier, who in turn uses a *verification key*, again provided by the same TTP, to assert whether the circuit computation was performed correctly.

While the various ZK-SNARK (or ZK-STARK) frameworks differ in the details, it is intuitive to see that the complexity of generating the proof (which dominates the cost of the protocol) must depend on the size of the circuit, which in turn depends on the amount of multiplications and additions performed in the computation: in the case of Merkle Tree commitment verification, most of the computation consists in iterating the underlying hash function. Since the finite field over which ZK-SNARK frameworks works is typically a huge prime field ($\approx 2^{256}$ elements), traditional hash functions like MD5 [10] or SHA [3], which are designed to be extremely efficient on classical boolean circuits, become extremely inefficient in the ZK case.

It is no wonder then, that in the last years researchers began to study so-called ZK-friendly cryptographic permutation (ZKFCP) designs that exploit the features of large prime fields to be efficient when translated into airhtmetic circuit, fundamentally resulting in a one-to-one mapping. Being a new research topic, these designs have seen a rapid series of improvements [1, 6, 5] in the last three years: in a two-part series of papers undergoing publication, we presented an algebraic framework, called *Generalized Triangular Dynamical System* (GTDS), which allows to express many of the existing cryptographic permutation designs and eases the construction of new ones, while at the same time giving strong security guarantees, and we then applied it to devise the `Blocc` blockcipher and the `Stamp` hash function. Using the `libsnark`[1] library (an implementation of the Groth16 framework), we implemented our hash function, along with other competitor hash functions and a hash-agnostic variable-arity Merkle Tree circuit template, in a `C++` project which we then used to compare their real-world performance for same-level security gaurantees in various scenarios.

Structure of the thesis

This work is organized in two parts: Part I contains the background of the work, presenting all the mathematical, computational and cryprographical tools and concepts required to understand the theory, the history and the applications of ZKFCPs. In Part II, we begin with a review of state of the art ZKFCPs, we then present the GTDS algebraic framework, its instantiation in the form of the `Blocc` block cipher and the `Stamp` hash function and we conclude with an implementation analysis and experimental comparison between the current SoTA and the GTDS constructions.

[1]`https://github.com/scipr-lab/libsnark`

I
Foundations

Zero Knowledge Proof (ZKP) systems are a relatively recent research topic: while the idea in itself, like many other beautiful ideas, is simple and elegant, its formalization, and even more so its realization, is all but trivial. A first rigorous description of what it means for a proof system to be *Zero Knowledge* was given by S. Goldwasser, S. Micali and C. Rackoff in 1985 [4] (the work was later updated in 1989).

To fully understand the properties of ZKP system, one needs to have an understanding of both fundamental and more advanced notions from the fields of group theory, computational theory and cryptographical theory. This is even more necessary for ZK-SNARK systems and ZK-friendly hash functions. For this reason, in this first part of the work we will (hopefully) give an exhaustive description of the tools required to have a better grasp of the results that will be presented in the second part.

2

Mathematical Background

In this chapter we will introduce all the mathematical concepts behind ZKP and ZK-friendly functions. While we decided, for completeness, to include even some of the more fundamental notions, we still expect the reader to have at least a rough idea of these concepts. Section 2.1 will introduce prime fields, cyclic groups other related notions.

2.1 Finite algebra

In algebra, a *tuple* consisting of one or more *sets* together with one or more *operations* over the sets is called an *algebraic structure*. Such structures can be organized according to a quite wide taxonomy, depending on whether they satisfy certain properties or not. We will denote sets with capital letters (e.g. S), a generic operation with a circled dot $\odot$ and algebraic structures with blackboard bold letters (e.g. $\mathbb{A}$). We will also denote elements of a set with lowercase letters $a, b, c, \ldots$ and variables over a set with lowercase letters $x, y, z, \ldots$. Finally, we will often use the term algebra to mean algebraic structure, whenever we belive the meaning to be clear from the context.

Remark 1. Some letters will be used reserved to denote some common algebraic structures. In particular, $\mathbb{B}$ will denote the boolean algebra, while $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ and $\mathbb{C}$ will denote, respectively, the natural, integer, rational, real and complex numbers.

We will denote the *cardinality* of set S with $|S|$, and use the same notation for the *arity* of an operation: for example, if $\odot$ is a binary operation, like integer addition[1], then $|\odot| = 2$. As a common abuse of notation in literature, when an algebraic structure $\mathbb{A}$ has exactly one *underlying set* A we will identify the two, e.g. by writing $x \in \mathbb{A}$ to mean $x \in A$.

For computer science applications, algebras with a finite number of elements play a central role.

Definition 1 (Finite algebra)**.** Given an algebraic structure $\mathbb{A}$, the *order* $|\mathbb{A}|$ denotes the number of elements inside (the underlying set of) $\mathbb{A}$. If $|\mathbb{A}| \in \mathbb{N}$, then $\mathbb{A}$ is a finite algebra.

Elements of differents algebraic structures can be associated through *morphisms*.

[1]if considered as a relation, addition would actually be ternary.

Definition 2 (Homomorphism)**.** Given two algebras $\mathbb{A} = (A, \odot_1, \dots, \odot_n)$, $\mathbb{A}' = (A', \odot'_1, \dots, \odot'_n)$ such that $\forall i \leq n \colon |\odot_i| = |\odot'_i| = a_i$, an homomorphism is a function $h \colon A \to A'$ such that:

$$\forall i \leq n, \forall x_1, \dots, x_{a_i} \in A \colon h(\odot_i(x_1, \dots, x_{a_i})) = \odot'_i(h(x_1), \dots, h(x_{a_i})) \qquad \text{(linearity)}$$

We say that $\mathbb{A}$ is homomorphic to $\mathbb{A}'$ through h.

Definition 3 (Isomorphism)**.** An isomorphism is an homomorphism whose inverse is also an homomorphism. Given two algebras $\mathbb{A}$ and $\mathbb{A}'$, if there exists some isomorphism $h \colon \mathbb{A} \to \mathbb{A}'$ we write $\mathbb{A} \cong \mathbb{A}'$.

Definition 4 (Endomorphism, Automorphism)**.** An endomorphism is a homomorphism from an algebraic structure $\mathbb{A}$ to itself. An automorphism is an endomorphism which is also an isomorphism.

2.1.1 Groups

We will now introduce some important classes of algebraic structures equipped with one fundamental operation.

Definition 5 (Monoid)**.** A monoid is a pair $\mathbb{M} = (M, \odot)$, where M is the underlying set and $\odot \colon M \times M \to M$ is the *composition* operation, such that the following properties are satisfied:

$$\begin{aligned} &\forall x, y \in M \colon x \odot (y \odot z) = (x \odot y) \odot z && (\textit{associativity}) \\ &\exists \mathrm{e} \in M \colon \forall x \in M \colon x \odot \mathrm{e} = x && (\textit{identity element}) \end{aligned}$$

$\mathbb{M}$ is a *commutative (or abelian) monoid*, if it also holds that:

$$\forall x, y \in M \colon x \odot y = y \odot x \qquad (\textit{commutativity})$$

Finally:

$$\forall x \in \mathbb{M}, \forall k \in \mathbb{N} \colon x^k = \begin{cases} \mathrm{e} & k = 0 \\ x^{k-1} \odot x & k > 0 \end{cases} \tag{2.1}$$

Definition 6 (Cyclic Monoid)**.** A cyclic monoid is a monoid $\mathbb{M} = (M, \odot)$ such that:

$$\exists g \in M \colon \mathbb{M} = \langle g \rangle = \left(\left\{ g^k \right\}_{k \in \mathbb{N}}, \odot \right)$$

Definition 7 (Group)**.** A group is a monoid $\mathbb{G} = (G, \odot)$, such that:

$$\forall x \in G \colon \exists x^{-1} \in G \colon x \odot x^{-1} = \mathrm{e} \qquad (\textit{inverse element})$$

With the notion of inverse element, we can rewrite and extend Equation (2.1) for groups as follows:

$$\forall x \in \mathbb{G}, \forall k \in \mathbb{Z} \colon x^k = \begin{cases} x^{k-1} \odot x & k \geq 0 \\ x^{k+1} \odot x^{-1} & k < 0 \end{cases}$$

If $\mathbb{G}$ is also a commutative (resp. cyclic) monoid, then it is a commutative (resp. cyclic) group. A group $\mathbb{G}' = (G', \odot')$ is a *subgroup* of $\mathbb{G}$ if $G' \subseteq G$ and $\odot' \subseteq \odot$.

The identity element e of a monoid is typically denoted with 1 in numeric algebras, when the composition operation resembles standard multiplication, or by 0 when the composition operation resembles standard addition. We use the notation $\mathrm{e}_{\mathbb{A}}$ (or $1_{\mathbb{A}}$, $0_{\mathbb{A}}$) to specify the algebra over which we intend to pick the identity element, dropping the subscript when $\mathbb{A}$ is clear from the context.

It is important to stress that one should be careful not to confuse the symbol and the name of an operation or of a special element with its semantics: the syntax to denote the inverse element x^{-1} of a group is reminiscent of standard multiplication inversion, but this is not the case in general. When the composition operation resembles standard addition, the inverse is more likely denoted with $-x$.

Example 1. The algebra $\mathbb{A} = \mathbb{Z} \setminus \{\times\}$ (i.e. integer numbers without multiplication) is an abelian group: addition is associative and commutative, the identity element is $\mathrm{e}_{\mathbb{A}} = 0$, and every number x has an inverse $x^{-1} = -x$ (e.g. $42^{-1} = -42$).

Example 2. Given a commutative group $\mathbb{G} = (G, \odot)$, consider the algebra $\mathrm{End}(\mathbb{G})_+ = (H, +)$, where H is the set endomorphisms over $\mathbb{G}$ and $+\colon H \times H \to H$ is defined as: $(h_1 + h_2)(x) \equiv (h_1 + h_2)(x)$.

$\mathrm{End}(\mathbb{G})_+$ is a commutative group: $+$ is associative and commutative, the identity element is $\mathrm{e}_{\mathrm{End}(\mathbb{G})_+} = z$, where z is the zero endomorphism (i.e. $\forall x \in G\colon z(x) = \mathrm{e}_{\mathbb{G}}$); finally, every homomorphism $h \in H$ has an inverse $h^{-1} = -h$ such that $\forall x \in G\colon (-h)(x) = h(x)^{-1}$ (in this example, using the h^{-1} notation causes confusion with the inverse function!).

Example 3. Consider now the algebra $\mathrm{End}(\mathbb{G})_\circ = (H, \circ)$ where $\mathbb{G}$ and H are defined as in Example 2, and $\circ\colon H \times H \to H$ is function composition: $(h_1 \circ h_2)(x) \equiv h_1(h_2(x))$.

$\mathrm{End}(\mathbb{G})_\circ$ is a monoid: function composition is associative, and the identity element is $\mathrm{e}_{\mathrm{End}(\mathbb{G})_\circ} = \mathrm{id}$, where id is the identity endomorphism (i.e. $\forall x \in G\colon \mathrm{id}(x) = x$).

2.1.2 Fields

Many algebraic structures rely on two fundamental operations, called *addition* and *multiplication*: two important types of such structures are *rings* and *fields*.

Definition 8 (Ring)**.** A ring is a triple $\mathbb{O} = (O, \oplus, \otimes)$ where O is the underlying set, $\oplus\colon O \times O \to O$ is the *addition* operation and $\otimes\colon O \times O \to O$ is the *multiplication* operation, sukh that the following properties are satisfied:

$$\begin{aligned}
&\mathbb{O}_\oplus = \mathbb{O} \setminus \{\otimes\} \text{ is an abelian group} \\
&\mathbb{O}_\otimes = \mathbb{O} \setminus \{\oplus\} \text{ is a monoid} \\
&\forall x, y, z \in O\colon x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z) && \textit{(left distributivity)} \\
&\forall x, y, z \in O\colon (y \oplus z) \otimes x = (y \otimes x) \oplus (z \otimes x) && \textit{(right distributivity)}
\end{aligned}$$

If $\mathbb{O}_\otimes$ is a commutative monoid, then $\mathbb{O}$ is a *commutative (abelian) ring.*

Given a ring $\mathbb{O}$ and an element $x \in \mathbb{O}$, we denote its inverse w.r.t. addition as $-x$, while maintaining the notation x^{-1} for the multiplicative inverse. Furthermore, the identity element w.r.t. addition, denoted $\mathrm{e}_\oplus$, will also be denoted as 0, while the identity element w.r.t. multiplication, denoted $\mathrm{e}_\otimes$, will maintain its alternative notation as 1.

Definition 9 (Field)**.** A field is a commutative ring $\mathbb{F} = (F, \oplus, \otimes)$ such that $0 \neq 1$ and $\mathbb{F}_\otimes \setminus \{0\}$ is a commutative group.

Fields are one of the most important and studied algebraic structures: the algebra of real numbers $\mathbb{R}$ is a field, as is the algebra of complex numbers $\mathbb{C}$.

Remark 2. Given the set of integers $Z_q = \{0, \dots, q-1\}$, we denote with $\oplus_q$ integer sum modulo q, and with $\otimes_q$ integer multiplication modulo q. The algebra $\mathbb{Z}_q = (Z_q, \oplus_q, \otimes_q)$ is a finite ring for any $q \in \mathbb{N}$, and it is a finite field if and only if q is prime.

Example 4. Boolean circuits with XOR and AND gates behave like elements of the boolean field $\mathbb{B} = (\{\bot, \top\}, \textsc{xor}, \textsc{and})$. It is easy to show that $\mathbb{B} \cong \mathbb{Z}_2$. Similarly, k-bit unsigned integers sum and multiplication work as in $\mathbb{Z}_{2^k}$.

Example 5. Given an abelian group $\mathbb{G}$, the algebra $\mathbb{H}_\mathbb{G} = \mathrm{End}(\mathbb{G}) = \mathrm{End}(\mathbb{G})_+ \cup \mathrm{End}(\mathbb{G})_\circ$ is the *endomorphism ring* of $\mathbb{G}$: $\mathrm{End}(\mathbb{G})_+$ is an abelian group, $\mathrm{End}(\mathbb{G})_\circ$ is a monoid (cfr. Examples 2 and 3), and it is easy to show that $\circ$ distributes over $+$ both on the left and the right.

2.1.3 Vector spaces

It is possible to naturally extend the underlying operations of a field over tuples of field elements to obtain a *vector space.*

Definition 10 (Vector space)**.** A vector space is a quadruple $\mathbb{V} = (V, \mathbb{F}, +, \odot)$ where V is the underlying vector set, $\mathbb{F}$ is the underlying field, $+\colon V \times V \to V$ is the *vector addition* operation and $\odot\colon \mathbb{F} \times V \to V$ is the *scalar multiplication* operation, such that $\mathbb{V}_+ = (V, +)$ is a commutative group, and $\odot$ is an homomorphism between $\mathbb{F}$ and $\mathrm{End}(\mathbb{V}_+)$.

3 Computational Background

3.1 Turing Machines

4

Cryptographical Background

4.1 Hash functions

4.2 Verification Trees

4.3 Zero Knowledge Proofs

4.4 ZK-SNARK systems

II
Zero Knowledge friendly permutations

5 State of the art

5.1 MiMC

5.2 Poseidon

5.3 Griffin

5.4 Other designs

6

Generalized Dynamic Triangular Systems

7

Implementations and experiments

III

Appendix

A

Titolo della prima appendice

Sed purus libero, vestibulum ut nibh vitae, mollis ultricies augue. Pellentesque velit libero, tempor sed pulvinar non, fermentum eu leo. Duis posuere eleifend nulla eget sagittis. Nam laoreet accumsan rutrum. Interdum et malesuada fames ac ante ipsum primis in faucibus. Curabitur eget libero quis leo porttitor vehicula eget nec odio. Proin euismod interdum ligula non ultricies. Maecenas sit amet accumsan sapien.

Bibliography

[1] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 191–219, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[2] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

[3] Quynh H. Dang. *Secure Hash Standard.* National Institute of Standards and Technology, Jul 2015.

[4] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[5] Lorenzo Grassi, Yongling Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. A new feistel approach meets fluid-spn: Griffin for zero-knowledge applications. *IACR Cryptol. ePrint Arch.*, 2022:403, 2022.

[6] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security Symposium*, 2021.

[7] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. `https://eprint.iacr.org/2016/260`.

[8] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford University, Stanford, CA, USA, 1979. AAI8001972.

[9] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. Cryptology ePrint Archive, Paper 2013/279, 2013. `https://eprint.iacr.org/2013/279`.

[10] Ronald L. Rivest. The md5 message-digest algorithm. In *RFC*, 1990.