D5.4 Ad-hoc BulletProof Statements

Grant Agreement nr	856879
Project acronym	PRESENT
Project start date (duration)	September 1st 2019 (36 months)
Document due:	August 31st 2021
Actual delivery date	August 31st 2021
Leader	UPF
Reply to	sergi.rovira@upf.edu
Document status	Submission version

Project funded by H2020 from the European Commission

Project ref. no.	856879
Project acronym	PRESENT
Project full title	**P**hotoreal **RE**altime **S**entient **ENT**ity
Document name	Ad-hoc BulletProof Statements
Security (distribution level)	Public
Contractual date of delivery	August 31st 2021
Actual date of delivery	August 31st 2021
Deliverable name	D5.4 - Ad-hoc BulletProof Statements
Type	Report
Status & version	Submission Version
Number of pages	14
WP / Task responsible	UPF
Other contributors	ICERT
Author(s)	Sergi Rovira, Vanesa Daza
EC Project Officer	Ms. Diana MJASCHKOVA-PASCUAL Diana.MJASCHKOVA-PASCUAL@ec.europa.eu
Abstract	The goal of this document is to provide a protocol based on a type of zero-knowledge arguments called Bulletproofs to generate proofs for statements relevant to PRESENT. We do not only provide a single protocol but two efficient libraries that can be used to prove a wide variety of statements. Moreover, we introduce a new use-case scenario where we use PRESENT virtual agents to provide security and trust based on the capabilities of Bulletproofs.
Keywords	Security module, authentication, registration.
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	Reason for Change
0.0 01-07-2021	Document created by UPF
1.0 26-07-2021	Version for internal review
2.0 31-08-2021	Submission Version

Table of Contents

1. EXECUTIVE SUMMARY

PRESENT is a three-year EU Research and Innovation project involving 8 companies and research institutions that are collaborating to create virtual digital humans (virtual agents) that look entirely naturalistic, demonstrate emotional sensitivity, establish an engaging dialogue, add sense to the experience, and act as trustworthy guardians and guides in the interfaces for AR, VR, and more traditional forms of media. These virtual agents can be used for interaction with human users in several scenarios.

The project includes a security component whose main goal is that of authenticating the parties. Human users are equipped with digital identities (basically a digital identity card) and can use such identities to prove who they are through a face recognition authentication that compares the face of the user with the face stored in the digital identity card.

The design of the security module has been described in deliverable D5.2 *Trust and security infrastructure design*. The key technology that is used to implement the module is the Sovrin Blockchain, which allows us to handle credentials (digital identities) in a decentralized manner. The motivations for the use of the Sovrin framework have been discussed in the deliverable D5.1 *Blockchain Privacy Report*.

The goal of this document is to provide a protocol based on a type of zero-knowledge argument called Bulletproofs to generate proofs for statements relevant to PRESENT. We do not only provide a single protocol but two efficient libraries that can be used to prove a wide variety of statements. Moreover, we introduce a new use-case scenario where we use PRESENT virtual agents to provide security and trust based on the capabilities of Bulletproofs.

2. INTRODUCTION

The main goal of this document is to explain how a particular type of cryptographic scheme can be used to prove particular statements and provide extra functionality for the benefit of the PRESENT project.

With the Covid-19 pandemic, the need for virtual alternatives to services has become apparent. At the time of writing, most of Europe has begun or is in the process of relaxing measures as the vaccination process progresses at a steady pace. This means that events such as concerts, football matches, motor competitions, etc will be or are already allowed to be organized again. The use-case presented in this document can minimize human interaction and contact, reducing human error and providing high confidence for both organizers and spectators of such events.

The main idea is to use a PRESENT virtual agent as a virtual clerk which will take on the job of validating the tickets provided by the users at the entrance of the events. This drastically reduces the risk of infection for both spectators and the staff of the events. Moreover, the system could also validate that the attendees of the event have a Covid-19 green pass without them having to disclose any personal information, providing an extra layer of security. Without taking into account the pandemic, this use case is also highly relevant since malicious behaviour and human error on both sides is reduced to almost zero, improving confidence and trust for both organizers and clients.

The technology that we will use to implement this use case is a type of cryptographic system called zero-knowledge proof. In particular, we will use Bulletproofs, which are efficient instantiations of zero-knowledge arguments that require only one round of communication and no trusted setup.

The document is structured as follows. In Section 1 we introduce the relevant theoretical concepts in an intuitive way in order to equip the reader with the necessary background. In Section 2 we focus on the definition of the use-case and explore its potential. Finally, in Section 3 we give the details of the provided implementation, explaining its capabilities, functionalities, and efficiency.

3. BULLETPROOFS

The goal of this section is to give the reader an intuitive understanding of Bulletproofs. We begin with a very high-level description of zero-knowledge proofs and some of their categorizations. We will only explain the necessary concepts and definitions needed to understand Bulletproofs, which will be presented in the second part of this section.

3.1. An introduction to Zero-Knowledge Proofs

Zero-knowledge proofs (ZK) allow us to prove that some statement is true without revealing anything else apart from this fact. For example, we might want to prove that we are over a certain age without revealing our date of birth or that we are solvent without showing our portfolio.

The two players in a ZK proof are the *prover* (who wants to prove the statement) and the *verifier* (who wants to check if the statement is true).

A proof system needs to satisfy the following properties (stated very informally):

- **Completeness:** If the statement is true and both prover and verifier are honest (they follow the protocol specifications) the verifier will be convinced that the statement is true.

- **Soundness:** If the statement is false and the verifier is honest, the verifier will not be convinced that the statement is true.

- **Zero-knowledge:** If the statement is true and the prover is honest, the verifier will not learn any confidential information from the interaction with the prover, it will only learn the fact that the statement is true.

There is an important remark to be made regarding the soundness property. Notice that the definition of soundness provided above does not specify the computational power of the prover. If the prover does not have any limits on computational power, we simply talk of zero-knowledge proofs. However, when we limit the computational power of the prover to be small, we talk about *zero-knowledge arguments*. In other words, an argument is a proof that holds only if the prover is computationally bounded.

To prove some claims we require additional information called *instance* together with the confidential information which is called *witness*. The instance is known to both the prover and the verifier, but the witness is only known by the prover and must not be leaked during the interaction.

In order to provide the reader with some intuition, we give a few examples of zero-knowledge proofs where we present different scenarios, the statement that the prover wants to prove and the corresponding instances and witnesses.

Scenario	**Statement being proven**	**Instance**	**Witness**
Age above 18	I am an adult	Tamper-resistant identification chip	Birthdate and personal data (signed by a certification authority)

Solvency	I am not bankrupt	Encrypted & certified bank records	Portfolio data and decryption key
Asset ownership	I own this asset	A blockchain or other commitments	Sequence of transactions and secret keys that establish ownership
Chessboard configuration	This configuration can be reached	The rules of Chess	A sequence of valid chess moves

Table 1: Some examples of statements, instances and witnesses of ZK proofs.

Zero-knowledge proofs can be further divided into two categories: *interactive* and *non-interactive*. The former requires the prover and the verifier to communicate over a number of rounds bigger than one, while in the latter the prover sends a single message to the verifier, and no more communication between them is required.

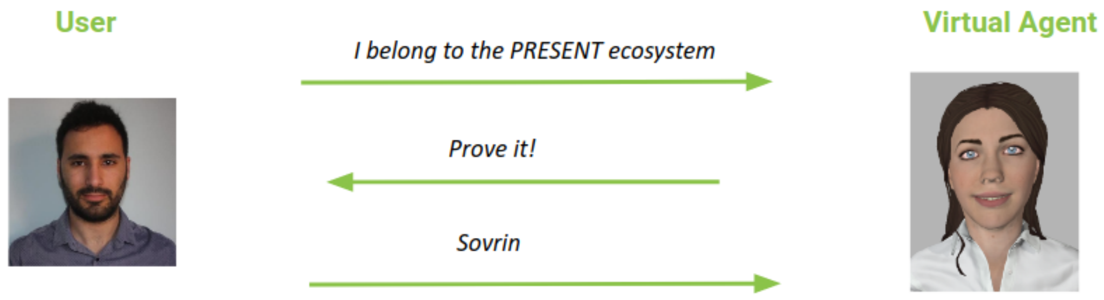


Figure 1: Example of interaction between a prover and a verifier. The last message sent by the prover depends on the specific proof system used.

In the next section, we describe the main focus of this document: Bulletproofs. They are non-interactive zero-knowledge proofs first designed to provide both efficiency and confidentiality to cryptocurrencies. The main reason for choosing Bulletproofs for PRESENT is the fact that they do not require a trusted setup and they are efficient enough for our purposes.

3.2. The ideas at the core of Bulletproofs

Bulletproofs are short *non-interactive zero-knowledge arguments of knowledge* that require no trusted setup. This means that the prover sends a single message to the verifier and that they do not need to rely on any form of prior information generated by a trusted third party. Moreover, the computational power of the prover is bounded while the verifier might have unlimited computational resources.

Bulletproofs were designed to enable efficient confidential transactions in cryptocurrencies, but have found many other applications, such as shortening proofs of solvency or enabling confidential smart contracts [1].

The technicalities and mathematics background necessary to understand Bulletproofs are well beyond the scope of this document. For the interested reader, good technical references are [1,4]. In this document, we only give a very high-level introduction to the main ideas of Bulletproofs.

We start by discussing a cryptographic primitive which is at the core of almost all modern cryptographic constructions, commitment schemes. A *commitment scheme* allows us to select a secret value and commit to it in the sense that the party performing the commitment can not change that value for another in the future. The scheme gives the capability to reveal the value later on, but this is not a mandatory task. The idea of a commitment scheme is the digital-analog of a sealed envelope.

The main functionality of Bulletproofs is to prove that a committed value lies within a certain interval. For example, in the context of blockchains, it is very useful to have an efficient protocol to prove that a secret value lies in the interval [0,2^n] for some large value of n. In the cryptographic community, this functionality is called an *aggregatable* range proof.

Range proofs allow us to prove that a secret value (previously encrypted or committed to) lies in a certain interval. They do not leak any information about the secret value but the fact that it lies within the desired interval. Now, consider the case where a prover needs to provide multiple range proofs at the same time[1]. The idea of *aggregated range proofs* is to build a system that can provide a proof for multiple secret values and its efficiency is better than doing one proof for each of the secrets.

Bulletproofs are often compared to another very popular cryptographic scheme called *succinct non-interactive arguments of knowledge* (SNARKs). For our purposes, the main difference between the two cryptographic primitives is the fact that bulletproofs do not require trust, but SNARKs do. To keep the document at a reasonable level we do not provide any further explanation about this construction, the reader can refer to [5] for the technical details. Nevertheless, we do provide an implementation of the most efficient instantiation of SNARKs together with the implementation of Bulletproofs.

[1] For example, in the context of blockchains, this happens when a transaction has multiple outputs.

The main use-case for Bulletproofs is providing confidentiality in blockchain transactions. For us, these types of ZK proofs will provide many applications that could be beneficial to PRESENT in the future. One of which is the use-case described in the next section.

4. A NEW USE-CASE

We can use Bulletproofs to provide short and efficient proofs of ownership which will be verified by a PRESENT virtual agent. That is, in our context, the present virtual agent plays the role of the verifier and the user the role of the prover.

Imagine that our virtual agent is instantiated as a receptionist in a hotel. We have previously booked a room for the night and the hotel has issued a ticket. Bulletproofs can be used to check that we indeed own this ticket. Another scenario is that of a concert. We have our agent instantiated as a receptionist in a concert hall. We use Bulletproofs to check that we own the matching ticket, and we are allowed to enter.

We could extend this idea to sports events, conferences, theatres, museums, airports etc.

The main benefit of using a PRESENT virtual agent versus a human employee is the removal of human error, providing a high degree of confidence to companies and customers. Moreover, the COVID-19 pandemic has provided a further incentive to implement this technology as soon as possible. Having a virtual agent taking the task of validating tickets removes the danger of infection between employees and customers.

Another benefit of using a virtual agent over traditional methods is efficiency. Verifying a proof is a much faster system, requiring only the scanning of the ticket and the verification computation, which can take only a few milliseconds in total. For a reference, our implementation can verify a proof of 64 bits in just a few milliseconds.

The system could also be used to provide other services such as taking the temperature of the attendees or checking multiple tickets at once (for groups of people for example) in just a few milliseconds.

5. IMPLEMENTATION DETAILS

In this section, we provide the capabilities and implementation details of the libraries that we have created to instantiate the use-cases mentioned above. It is important to remark that these libraries are not integrated within PRESENT at the moment, and they will not be integrated into it during the duration of the project.

We implemented Bulletproofs as a module integrated into *ZPiE*[2], a Zero-Knowledge Proofs library coded in C. The library uses GMP and MCL as dependencies: GMP is a pure C library used to handle big numbers and operations involving them, and MCL is a library written in C++, which offers a C wrapper for using it in pure C projects, used to do elliptic curve operations. The library also supports the elliptic curves BN128 and BLS12-381, which are thus also supported by our implementation. We implemented an API which allows us to generate aggregated range proofs using the Bulletproofs scheme above referred, and to verify them. The instructions on how to compile and use the library can be found in the README of the repository.

We benchmarked our implementation as depicted in Figure 1. Moreover, we improved the efficiency of our solution by using multi-threading in several parts of either the prover and the verifier, where splitting the operations in different cores was possible. As we can see, we are benchmarking the time it takes by the prover and the verifier, either in single-core (SC) or multi-core (MC) to compute the proofs and verify them. We performed the experiments for several amounts of aggregated proofs of 64 bits, using a 4-cores CPU, and the BLS12-381 curve (which offers 128 bits of security).

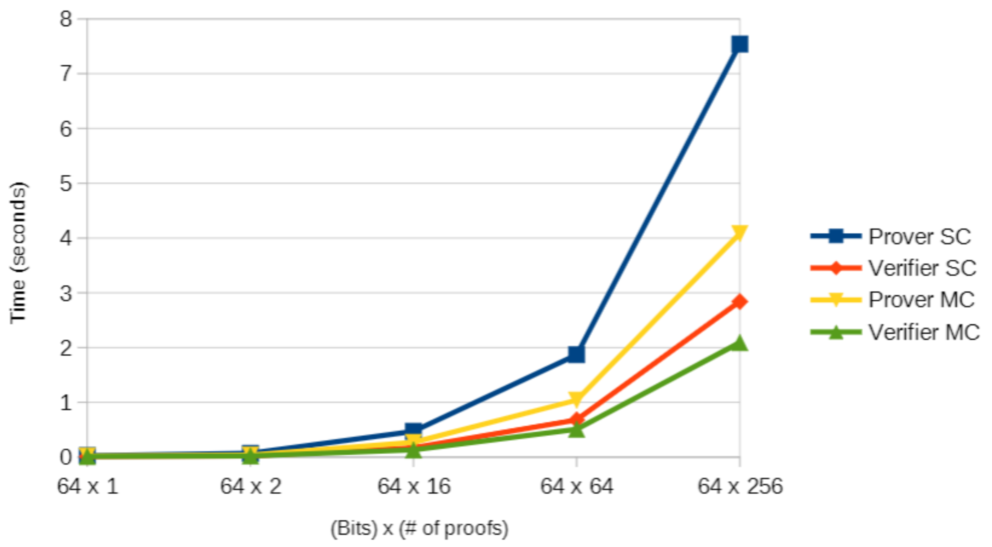


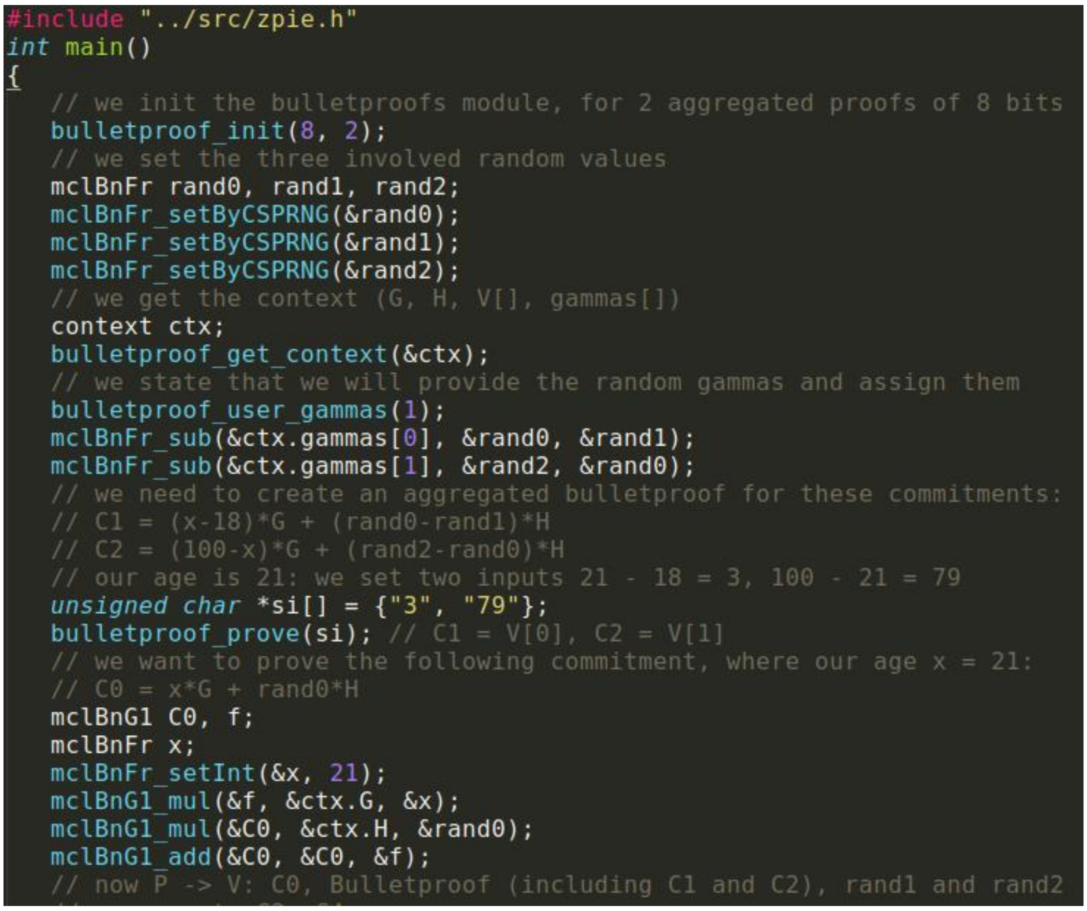
Figure 1: ZPiE Bulletproofs benchmarks using an Intel Core i7-11370H @ 3.30GHz

As a concrete example, we will show how we can prove that we are over 18 years old using our software. Notice that this boils down to showing that our age is a natural number in the range [18-100] and, as we have mentioned in the previous section, this is indeed what bulletproofs allow us to prove very efficiently. For our example we will assume that

[2] You can find the latest version in this repository: https://github.com/xevisalle/zpie.

we are 21 years old. We show how this would be done in our library in figures 2 and 3[3]. As per performance, doing this proof takes around 30 to 40 milliseconds on average.

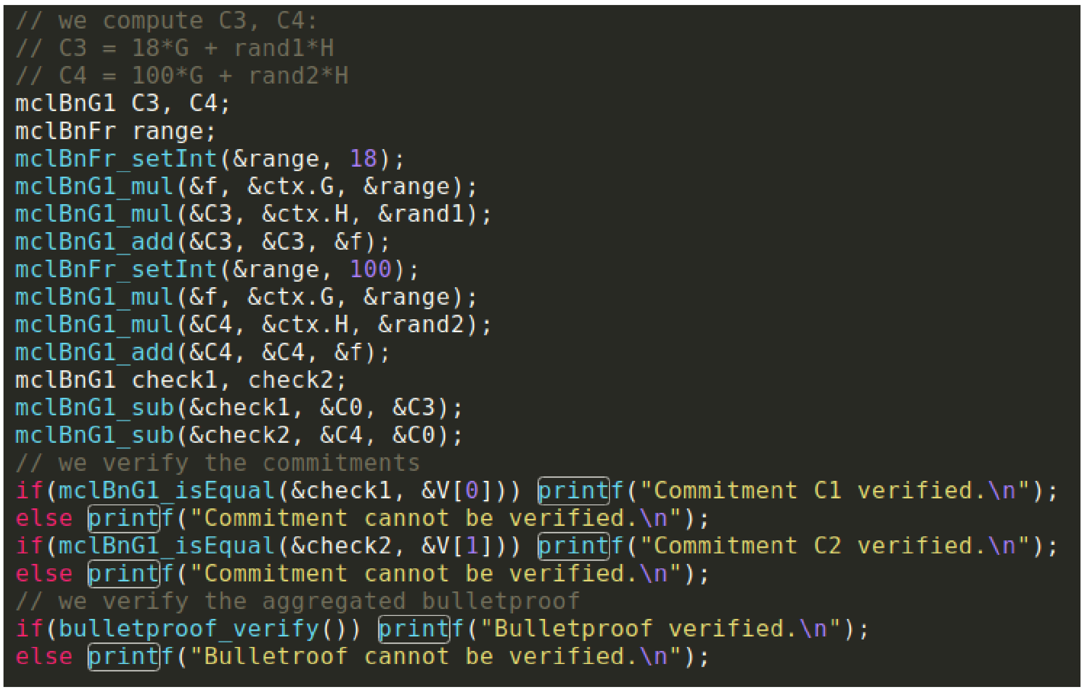
We want to efficise that the above example only requires one proof. We could easily extend it to the scenario where we need to prove that everyone in a particular group is over 18 years old. In this case, we could take advantage of the efficiency results shown in Figure 1.



```
#include "../src/zpie.h"
int main()
{
   // we init the bulletproofs module, for 2 aggregated proofs of 8 bits
   bulletproof_init(8, 2);
   // we set the three involved random values
   mclBnFr rand0, rand1, rand2;
   mclBnFr_setByCSPRNG(&rand0);
   mclBnFr_setByCSPRNG(&rand1);
   mclBnFr_setByCSPRNG(&rand2);
   // we get the context (G, H, V[], gammas[])
   context ctx;
   bulletproof_get_context(&ctx);
   // we state that we will provide the random gammas and assign them
   bulletproof_user_gammas(1);
   mclBnFr_sub(&ctx.gammas[0], &rand0, &rand1);
   mclBnFr_sub(&ctx.gammas[1], &rand2, &rand0);
   // we need to create an aggregated bulletproof for these commitments:
   // C1 = (x-18)*G + (rand0-rand1)*H
   // C2 = (100-x)*G + (rand2-rand0)*H
   // our age is 21: we set two inputs 21 - 18 = 3, 100 - 21 = 79
   unsigned char *si[] = {"3", "79"};
   bulletproof_prove(si); // C1 = V[0], C2 = V[1]
   // we want to prove the following commitment, where our age x = 21:
   // C0 = x*G + rand0*H
   mclBnG1 C0, f;
   mclBnFr x;
   mclBnFr_setInt(&x, 21);
   mclBnG1_mul(&f, &ctx.G, &x);
   mclBnG1_mul(&C0, &ctx.H, &rand0);
   mclBnG1_add(&C0, &C0, &f);
   // now P -> V: C0, Bulletproof (including C1 and C2), rand1 and rand2
```

Figure 2: Setup and initialization of the aggregated proof using ZPIE

[3] We have provided comments within the code to make it easier to understand. We refer the reader to the github repository (https://github.com/xevisalle/zpie) for more details.



```c
// we compute C3, C4:
// C3 = 18*G + rand1*H
// C4 = 100*G + rand2*H
mclBnG1 C3, C4;
mclBnFr range;
mclBnFr_setInt(&range, 18);
mclBnG1_mul(&f, &ctx.G, &range);
mclBnG1_mul(&C3, &ctx.H, &rand1);
mclBnG1_add(&C3, &C3, &f);
mclBnFr_setInt(&range, 100);
mclBnG1_mul(&f, &ctx.G, &range);
mclBnG1_mul(&C4, &ctx.H, &rand2);
mclBnG1_add(&C4, &C4, &f);
mclBnG1 check1, check2;
mclBnG1_sub(&check1, &C0, &C3);
mclBnG1_sub(&check2, &C4, &C0);
// we verify the commitments
if(mclBnG1_isEqual(&check1, &V[0])) printf("Commitment C1 verified.\n");
else printf("Commitment cannot be verified.\n");
if(mclBnG1_isEqual(&check2, &V[1])) printf("Commitment C2 verified.\n");
else printf("Commitment cannot be verified.\n");
// we verify the aggregated bulletproof
if(bulletproof_verify()) printf("Bulletproof verified.\n");
else printf("Bulletroof cannot be verified.\n");
```

Figure 3: Computation and verification of a range proof using ZPiE

6. CONCLUSION

This deliverable provides details about Bulletproofs and how they can be used in the context of PRESENT.

We have provided all the necessary background on this technology and explained in detail our idea of using virtual agents to be instantiated as virtual clerks, increasing safety and trust for both providers and customers.

The document also describes in detail the libraries that we have implemented in order to put this technology in place.

7. REFERENCES

1. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille and G. Maxwell, "Bulletproofs: Short Proofs for Confidential Transactions and More," *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 315-334, doi: 10.1109/SP.2018.00020.

2. Damgård, I. "Commitment Schemes and Zero-Knowledge Protocols." *Lectures on Data Security*, 1998.

3. D. Boneh, V. Shoup, "A Graduate Course in Applied Cryptography", version 0.5, 2020, updated version at: https://toc.cryptobook.us/.

4. ZKProof. ZKProof Community Reference. Version 0.2. Ed. by D. Benarroch, L. T. A. N. Brandão, E. Tromer. Pub. by zkproof.org. Dec. 2019. Updated versions at https://zkproof.org.

5. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. "Succinct non-interactive zero knowledge for a von Neumann architecture". In Proceedings of the 23rd USENIX conference on Security Symposium (SEC'14). USENIX Association, USA, 781–796.