Bulletproofs With Stochastic Equation Sets

Michael Brand
School of Computing Technologies, RMIT University
Melbourne, Vic, Australia
INCERT GIE
Leudelange, Luxembourg
michael.brand@rmit.edu.au

Benoît Poletti
Luxembourg delegate to ICAO
TAG/TRIP, IATA CAWG & UNECE WP.29
INTERPOL
Global Cybercrime group member
INCERT GIE
Leudelange, Luxembourg
bpoletti@incert.lu

ABSTRACT

Bulletproofs are general-purpose Zero Knowledge Proof protocols that allow a Prover to demonstrate to a Verifier knowledge of a solution to a set of equations, represented as a Rank 1 Constraint System.

We present a protocol extending the standard Bulletproof protocol, which introduces a second layer of interactivity to the protocol, by allowing the Verifier to choose the set of equations after the Prover has already committed to portions of the solution.

We show that such Verifier-chosen (or stochastically-chosen) equation sets can be used to design smaller equation sets with less variables that have the same proving-power as their larger, deterministic counterparts but are, in practice, orders of magnitude faster both in proof generation and in proof verification, and even reduce the size of the resulting proofs. We demonstrate this with an example from a real-world application.

Our method maintains all the classical benefits of the Bulletproof approach: efficient proof generation, efficient proof checking, extremely short proofs, and the ability to use Fiat-Shamir challenges in order to turn an interactive proof into a non-interactive proof.

KEYWORDS

zero-knowledge proof, bulletproof, stochastic equation set

1 INTRODUCTION

In his seminal 1979 paper, Yao [28] observed that communication complexities can be reduced for a proof if the proof only requires probabilistic verification. A classic example is determining equality between two strings of bits. If Alice has the string $a_1, \dots, a_n$ and Bob has the string $b_1, \dots, b_n$, it takes $\Omega(n)$ bits of communication in order to determine deterministically that $\forall i, a_i = b_i$, but reaching arbitrary levels of confidence for this fact only requires $\Theta(1)$ bits to be communicated, and this through the following algorithm and repetitions thereof.

Let $r_1, \dots, r_n$ be a string of random bits jointly known to both Alice and Bob, then checking if $\sum_i r_i a_i = \sum_i r_i b_i$, which requires only 1 bit of communication, provides 1 bit of verification for the equality.

YYYY(X), 1–11

https://doi.org/XXXXXXX.XXXXXXX

This technique does not actually require the variables to be bits. The values a_i, b_i and r_i can, for example, belong to any finite field, $\mathcal{F}$, in which case the protocol requires $O(\log|\mathcal{F}|)$ bits of communication, and verifies equality with only a $\frac{1}{|\mathcal{F}|}$ probability of error without any need for repetitions. The method continues to work, because in essence it merely reduces the checking of n equalities to the checking of a single equality by taking a random linear combination of all n original equalities.

This observation by Yao has led to the development of the field of interactive proof systems [25], probabilistically checkable proofs [2, 3], and ultimately also zero knowledge proofs (ZKP) [16], proofs in which a Prover can convince a Verifier of a claim, without contributing anything to the knowledge of the Verifier other than the veracity of the claim itself. Today, we know [5, 6] that any statement in NP [12] can be proved in zero knowledge, even if the Verifier is only allowed to contribute random bits, as in Arthur-Merlin protocols [4].

One example of such a general-purpose ZKP system is Bulletproofs [11]. Bulletproofs have the distinction of allowing the Prover to convince the Verifier with uncommonly short proofs. The reduction in the necessary amounts of communication used in Bulletproofs is through the application of many sophisticated techniques, including several different incarnations of the frugal equality checking algorithm described above.

In the literature, this equality checking algorithm often comes in one of two variations, depending on where the random bits come from, and Bulletproofs utilise both techniques.

The first possibility is to rely on a third party to generate *public randomness*. Let $\mathcal{G}$ be an additive group with a prime size $|\mathcal{G}|$, and let $\mathcal{F}$ be the field of integers modulo $|\mathcal{G}|$. Bulletproofs require a group such as $\mathcal{G}$ for which there is a one-way function providing a homomorphic mapping from $\mathcal{F}$ to $\mathcal{G}$, and where given group elements $g_1, \dots, g_m$ it is cryptographically difficult to find a nontrivial linear combination of the elements that sums to zero (the group's neutral element).

Given such a group, it is possible for a trusted third party to generate random group elements $g_1, \dots, g_n \in \mathcal{G}$, at which point the Prover can simultaneously prove that n equations of the form $a_i = b_i$, over $a_1, \dots, a_n, b_1, \dots, b_n \in \mathcal{F}$, all hold, by merely proving the single equation $\sum_{i=1}^n g_i a_i = \sum_{i=1}^n g_i b_i$.

The second method, also used extensively in Bulletproofs, is *Reed-Solomon fingerprinting* [22]. This requires only a single random scalar $x \in \mathcal{F} \setminus \{0\}$, which is typically provided by the Verifier, to compress n equalities $a_i = b_i$, over $a_1, \dots, a_n, b_1, \dots, b_n \in \mathcal{G}$ into a single one: $\sum_{i=1}^n x^i a_i = \sum_{i=1}^n x^i b_i$ (noting that, unlike in the first technique, here the original equations were in $\mathcal{G}$, but the randomness in $\mathcal{F}$). Reed-Solomon fingerprinting works because equality can only occur if x is a zero of the polynomial $\sum_{i=1}^{n} X^i(a_i - b_i)$. Unless this is the zero polynomial, it can only have at most n solutions, so the probability of x being one of them is bounded by $n/|\mathcal{G}|$.

Bulletproofs represent their underlying NP problem by use of a *rank 1 constraint system* (R1CS) [27]. The specific flavour of R1CS used by Bulletproofs includes m triplets of variables (L_i, R_i, O_i), each satisfying

$$L_i \times R_i = O_i, \qquad (1)$$

these product operations being the only nonlinear portion of the equation system. The rest of the constraint system is a set of linear equations in the variables L_i, R_i and O_i, of which there are $3m$ in total, as well as in an unconstrained number of additional variables, known as *external variables*.

Algorithmically, Bulletproofs begin by reducing all equations, through Reed-Solomon fingerprinting, to a single equation. Although this is an operation whose complexity is as the number of nonzero coefficients in the linear equation set, it typically takes only a small portion of the proof's run-time because Bulletproof equation sets tend to be sparse. The rest of the Bulletproof algorithm, typically comprising the majority of the proof's run-time, has a complexity proportional to the number of variables, $3m$, or $O(m)$. This number can be quite large, leading to slow execution speeds.

An important observation is that while Bulletproofs utilise a wide array of techniques that has been developed over the past almost 50 years in probabilistic, interactive proof systems, it ultimately does so in order to prove knowledge of a solution to a standard, deterministic equation set. In this paper, we introduce a technique that allows *stochastic equation sets* to be represented in the structure underlying the Bulletproof, this enabling—by reuse of the very same techniques—the crafting of proofs that run faster, and are even shorter.

One property of standard Bulletproofs that we will want to retain in our stochastic model is that the Verifier can only ever contribute random bits, never anything structured or withholding any private information. The reason for this is that this proof structure allows the Verifier challenges to be produced by the Fiat-Shamir heuristic [15]. This is a key tool in the practical application of Bulletproof systems, because it allows such proofs to be made non-interactive. In our case, too, despite the stochastic nature of the proof, it will be possible to provide the proof in a noninteractive form.

1.1 Motivating example

There is a multitude of ways in which stochasticity in one's equation design can be used to make one's zero-knowledge proofs more compact. In this section, we outline one such technique in order to provide a concrete example, but it should be understood that this is only one method out of many to reap the benefits of stochasticity. The specific technique outlined here was used in a real-world solution to provide a 4-fold performance improvement to a Bulletproof implementation. We provide the full details of the real-world implementation, its optimisation, and the resulting performance improvements, in Appendix A. In this section, we describe the general technique employed, which can easily be adapted to other scenarios.

Consider that rank 1 constraint systems represent their equation sets algebraically, using ring operations (addition, subtraction and multiplication). This is a universal representation because if we merely store in each variable a value that is either zero or one then the addition and multiplication gates of the algebraic representation translate to logical OR and AND gates, and the entire equation set can be thought of as representing a general Boolean circuit.[1] However, in order to reduce the number of variables used and speed up the proofs, proof designers often explicitly utilise the algebraic properties of the equation set to their advantage, storing in each variable a value that is a large number, representing many bits of information.

The method to use stochasticity that is outlined in this section is beneficial to equation-set designers wishing to take the use of the algebraic properties of the R1CS representation to extremes, by attempting to reproduce an entire Diophantine equation set as a Bulletproof R1CS.

The problem with such an approach is that while the Diophantine equation set can be arranged so as to have exactly the same form as a Bulletproof R1CS, the two are not identical, and can have vastly different solution sets. The reason for this is that Bulletproofs do not work over integers but rather over $\mathcal{F}$, the integer residual group modulo $|\mathcal{G}|$. A modular equation set can have many solutions that are not also solutions to the integer equation set. Such spurious solutions are problematic, because the Prover may be able to convince the Verifier that they know a solution to the modular equation set without knowing a solution to the integer equation set.

Indeed, even when an integer solution does correspond to a modular solution, it is not uniquely determined by it, so even knowing a "valid" modular solution does not suffice to convince a Verifier that the Prover knows an integer solution.

While it is not possible to address this problem in the fully general case, it is possible to do so if the integer solutions can be bound in some way. If the integers in the solution can be bound to be of at most k bits, for example, one standard way to resolve the issue is to define k additional variables for each k-bit integer, these representing the bits of the larger integer variable.

For example, if a value A can be represented as $A = \sum_{i=0}^{k-1} 2^i a_i$, then (non-modular) integer equations over A can be represented via (modular) equations over the a_i.

Beyond simply reverting back to a Boolean circuit, this introduction of binary variables allows an equation-set designer to ascertain that integer variables are within a designated range, e.g. for the purpose of ensuring that certain operations do not result in overflows.

For example, if a variable A can be represented as $A = \sum_{i=0}^{k-1} 2^i a_i$, with the individual a_i all satisfying $a_i(a_i - 1) = 0$ so as to ensure that they are either 0 or 1, then this variable is known to be in the range $0 \le A < 2^k$. If k is chosen such that $2^{2k} < |\mathcal{G}|$, then multiplying two such variables by each other is guaranteed not to wrap around in modulo $|\mathcal{G}|$, so the result is the same as though the multiplication was in non-modular integer arithmetic. Altogether,

[1]Using $1 - x$ for negation, and $x(1 - x) = 0$ to ensure that inputs are all, indeed, zeros or ones.

such range bounding allows one to create equation sets that act as Diophantine, even though they are ultimately tested on modular integers, with no danger for spurious solutions or for the information of the modular solution to not be representative of knowledge of an integer solution.

This technique is useful in modelling integer arithmetic even in the common scenario when the integers in question are much larger than $|\mathcal{G}|$.

It is possible, as an example, to emulate long integer addition and long integer products by explicitly modelling the entire computation as occurring in some base 2^k representation. Choosing a suitably small k, the individual base-2^k digits can be represented as variables, and all long addition and long multiplication operations can be represented explicitly using all required intermediate variables (with the size limits for each variable enforced by an additional bit-representation for each variable).

While this straightforward representation approach works, it tends to lead to unnecessarily bloated equation sets and to slow execution. An alternative representation method is to pre-select a particular set of values $\{P_j\}_{j=1}^r$, and to test the correctness of the equation modulo each one of the P_j. By the Chinese remainder theorem (CRT) [20], if the equation holds modulo each one of the P_j, then it is also true modulo $L = \text{lcm}(P_1, \dots, P_r)$. If the numbers N to be represented in this way are all in the range $0 \le N < L$ then this is enough to prove knowledge of a solution to the corresponding integer equation set.

The modular representation method requires less extra variables than the long-integer-arithmetic method, but does still require constructing modular representations of each of the equation sets' variables under each of the moduli. For example, if A is an integer value equal to $\sum_{i=0}^{k-1} a_i 2^i$, where all a_i have been constrained to be either 0 or 1, then

$$A' = \sum_{i=0}^{k-1} a_i (2^i \bmod P_j) \tag{2}$$

equals A in mod P_j. (It may not equal "A mod P_j", because its value may still be greater than P_j, but it is nevertheless bounded by the only marginally larger value kP_j, and therefore can also be used to bound result value magnitudes.) Once all such modular variables are introduced, any equation originally having the integer form

$$\sum_{i=1}^{m} r_i L_i + s_i R_i + t_i O_i + w_i = 0$$

can be reworked to the modular form

$$\sum_{i=1}^{m} r_i L'_i + s_i R'_i + t_i O'_i + w_i = d_i P_j,$$

where L'_i, R'_i and O'_i are computed from L_i, R_i and O_i as described in (2), to produce results equivalent to the originals modulo P_j but in a much smaller range.

Because the left hand side was not altered modulo P_j, the right hand side must still be divisible by P_j, hence why setting it to $d_i P_j$ works.

Given the smaller range of all variables involved, this new equation can be guaranteed to be satisfiable without overflows, which we can ascertain by limiting the size of d_i. This is done using the same bit-representation technique that has been used to limit all other variable sizes.

The remaining problem is that while d_i can be guaranteed to be small, it may not be positive. To avoid this, we can set

$$\tilde{d_i} = d_i - d_i^{\min}$$

$$\tilde{w_i} = w_i - d_i^{\min} P_j,$$

where $d_i^{\min}$ (a known negative integer) is the smallest value that can be taken by d_i.

Our final equation then becomes

$$\sum_{i=1}^{m} r_i L'_i + s_i R'_i + t_i O'_i + \tilde{w_i} = \tilde{d_i} P_j,$$

Where all variables can be bound in size and restricted to be non-negative, and where this entire equation can by this be guaranteed to hold modulo $|\mathcal{G}|$ without ever incurring any overflows, for any suitably-small choice of P_j.

This technique allows us to represent large integer calculations with a relatively small number of extra variables per tested modulus. However, if N is large the number of moduli that need to be tested may also be large, contributing overall to a substantial addition to the number of variables in the equation set and to the runtime.

The problem is that all such moduli have to be tested. The use of randomness in generating one's equation set, which is the tool we develop in this paper, provides a better alternative. It allows us to decide on a much smaller set of moduli, taken from some large pool, and to test only those.

In principle, if the original integer equation is incorrect, it will remain incorrect when evaluated under almost any choice of modulus. For example, if the left-hand side and the right-hand side differ by an offset of Δ, only a modulus that is a divisor of Δ will cause evaluation under this modulus to yield equality.

We would therefore like to be able to test only one modulus (or at most a handful of them), but to choose these stochastically out of a very large set, knowing that the probability for the chosen one to erroneously validate the tested equation is negligible.

This approach trades off the original, bulky equation set, where every modulus P_j is tested, with a much lighter equation set, involving only a handful of moduli, but one that requires the equations to be chosen stochastically, at run-time. It is the adding of this capability to the basic Bulletproof that is the goal of this paper.

For completion of the motivating example, however, it should be noted that the approach of using randomly chosen moduli requires a careful strategy for the choice of moduli to be tested. It is not enough, for example, to pick a random element out of the original $\{P_j\}$, because it is trivial to choose a value that will be 0 modulo all elements except one, lowering substantially the strength of the test.

It is similarly not optimal to choose uniformly a random element between 2 and the maximum possible modulus. The reason for this is that with high probability one may end up with, e.g., a smooth number [18], and it is possible for an adversarial Prover to choose a nonzero discrepancy Δ for their equation that will be zero modulo many smooth numbers.

However, it is possible to prove that a proper choice of a rough modulus [18] can provide adequate strength to the test. Briefly, for the equation discrepancy, Δ, to be a multiple of a modulus, P, it must divide by each of P's factors. Because P is rough, each such factor must be greater than some threshold T. For Δ to be resilient against many such P, it must have many such factors, but if it has k such factors, its value is at least T^k. We know that $\Delta < N$, so this puts an upper bound for k at $\log_T N$, thus limiting the number of moduli that may erroneously attest to the correctness of an incorrect integer equation.

Using this technique, far less variables are needed in the proof, and its execution is much faster.

To demonstrate this with a concrete example, we have implemented a Bulletproof with a stochastic equation set as an improvement, along the same lines as discussed here, over a previously-optimised standard Bulletproof.

We developed the original Bulletproof equation set that is used in this example for INCERT GIE (the Luxembourgian public agency in charge of, among other things, securing digital signatures in national identity documents) as part of a proof of concept for a Zero-Knowledge digital wallet for official government documents. The equation set in our implementation verifies multiple cryptographic elements privately, including a 4096-bit RSA signature.

The full details of the original Bulletproof equation set used, as well as of the stochastic optimisation of it, are provided in Appendix A, together with a full report regarding the improvements in execution time and other metrics when implementing the stochastic approach.

In summary, however, the stochastic approach reduced the number of multiplication gates needed from 65, 536 to 16, 384, a 4-fold reduction, and this led to a corresponding 4-fold reduction in both proof generation times and proof checking times, while still maintaining a false-acceptance probability of under 2^{-67}.

The bulk of the multiplication gates that remain after the optimisation are simply used to store the 15, 324 bits that form the different parts of the secret signature that requires verification, so, counting only the extra multiplication gates that were added for verification purposes (what, later on in the paper, we refer to as "proof variables"), the stochastic version reduced the number of these from $65,536 - 15,324 = 50,212$ to $16,384 - 15,324 = 1,060$, a reduction by a factor of over 47.

This demonstrates the practical applicability of the method.

The problem that we address in this paper is how to enable Bulletproofs to support this type of optimisation, where the choice of an equation set depends on earlier random choices, making the equation set itself stochastic. Clearly, it would have been possible to simply include the equations relevant to each of the possible moduli as part of the overall equation set, or, alternatively, to represent the modulus operations themselves in the language of R1CS. However, all such approaches would have yielded a massively bloated equation set, which is precisely what we are trying to avoid. Thus, the stochastic nature of the equation set is unavoidable, and lies at the heart of the issue we are trying to solve.

2 PRELIMINARIES

To begin with, let us recap the relevant parts of the inner workings of a Bulletproof.

As already mentioned, Bulletproofs use 3 vectors of m variables each: L_i, R_i and O_i, satisfying (1), with a linear set of equations connecting these variables, which can be described as

$$W_L L + W_R R + W_O O = W_v v + \vec{c}.$$

Here L, R and O are the vectors of all L_i, R_i and O_i variables, W_L, W_R, W_O and W_v are matrices of coefficients, $\vec{c}$ is the vector of constants used in the linear equations, and v is the vector of external variables. In Bulletproofs, m is required to be a power of 2.

A common use for zero knowledge proofs is to ascertain particular facts regarding the external variables without having to communicate the actual values of these variables. The way to do this in Bulletproofs is to begin the proof by cryptographically committing [10, 16] on the value of each external variable, v_i, and then to use the proof itself to ascertain regarding the value of each variable whatever is needed.

The rest of the variables are known as *internal variables.*

In standard Bulletproofs, both the Prover and the Verifier use random elements of the group $\mathcal{G}$ that have previously been supplied by a trusted third party. Let us name these elements B, $\tilde{B}$, Q, $\{G_i\}_{i=1}^m$ and $\{H_i\}_{i=1}^m$. The commitments on the external variables, occurring in the first step of the proof, take the form of Pedersen Commitments [13, 19]:

$$V_j = v_j B + \tilde{v}_j \tilde{B}.$$

Here, each $\tilde{v}_j$ is an independently and uniformly chosen random blinding factor. For commitment, the value V_j is communicated to the Verifier.

The internal variables L, R and O are also committed on in the first step of the proof, but unlike the external variables, they are not committed on individually. Instead, they are committed on jointly, using Pedersen vector commitments, as follows.

$$A_I = \langle L \cdot G\rangle + \langle R \cdot H\rangle + \tilde{a}\tilde{B},$$

$$A_O = \langle O \cdot G\rangle + \tilde{o}\tilde{B},$$

$$S = \langle s^L \cdot G\rangle + \langle s^R \cdot H\rangle + \tilde{s}\tilde{B}.$$

Here, $\langle\cdot\rangle$ indicates dot product, $\tilde{a}$, $\tilde{o}$, $\tilde{s}$, $\{s_i^L\}_{i=1}^m$ and $\{s_i^R\}_{i=1}^m$ are all independent, uniformly-chosen, random blinding variables, and A_I, A_O and S are the values communicated to the Verifier to form the commitment.

Only after these commitments does the Verifier make its first contribution to the protocol, this being a random choice of $y, z \in \mathcal{F}$ values which the Verifier sends to the Prover. These are used for Reed-Solomon fingerprinting, reducing the entire equation set to a single equation, which is then proved.

The mechanics of the rest of the Bulletproof are beyond the scope of this paper. Consider, however, the problems of introducing a stochastic equation set to this format.

The Verifier can introduce a new random challenge r that will ultimately be used to choose final equations to be verified, but the question is when can such an r be revealed. Because a different equation set requires a different set of solutions for the values of L, R and O, it is not possible for the Prover to commit to the solutions, using $\{V_j\}$, A_I, A_O and S, unless r is revealed first. However, if r is revealed prior to $\{V_j\}$, A_I, A_O and S, the Prover may come up with a set of solutions that only solves for the single equation set tested, rather than any equation set that could have possibly been chosen. By this the Prover will be able to claim a falsehood.

3 THE SOLUTION IDEA

For our purposes, we will require a different dichotomy of the variables used, rather than just dividing them to "external" and "internal". We divide the Bulletproof variables to *target variables* and *proof variables*. Zero knowledge proofs are proofs that particular secret values known to the Prover satisfy particular known equations without the Prover having to divulge the secret values. The variables holding these values regarding which we are trying to claim statements we call target variables. All external variables are target variables, but they do not have to be the only ones. Zero knowledge proofs can also be used as proofs of knowledge, in which case the Prover merely attests that they know a solution to an equation set without having to make an external commitment to that solution. Variables that represent a solution of this type are also target variables. Proof variables are all other variables used in the system. They are there in order to facilitate the mechanism of the proof itself, rather than be part of the proven statement.

The idea for supporting stochastic equation sets is to first commit to the target variables, then to receive the value r from the Verifier, and only then to commit to the proof variables. For example, returning to our motivating example, we might want to first commit to some or all of the variables that do not depend on the choice of a modulo, P_j, and to commit on the rest only after the choice of P_j is made.

Importantly, however, for stochasticity to be effective it must be introduced without this increasing the overall complexity of the proofs: we do not want, for example, to require additional public randomness, more variables or more equation coefficients.

To ensure this, our solution does not alter the original structure of the equation set, and therefore allows the entire rest of the Bulletproof to continue unchanged and at no additional cost. We make no change to the external commitments, either, all of which correspond to target variables, and all of which can be committed to immediately at the beginning of the protocol, before r is known.

Our algorithm is designed to be run immediately after the commitment on external variables. Its key is the introduction of a new form of internal commitments, which will replace the standard commitments on A_I, A_O and S, and allow the rest of the algorithm to proceed exactly as usual.

What we prove in this paper is that in calculating the new internal commitments using our technique, the Prover cannot alter their internal target variable commitments after the value of r is known.

4 THE ALGORITHM

The algorithm meeting all requirements discussed is presented in Algorithm 1.

This algorithm makes several underlying assumptions regarding the identity of the target variables. The assumptions are:

(1) All target variables are part of either L or R, not O.
(2) For all i, it is never the case that both L_i and R_i are target variables.
(3) Let W_L^T, W_R^T and W_O^T be the portions of the coefficient matrices W_L, W_R and W_O, respectively, that relate to target variables (each variable's coefficients being a column), then (W_L^T, W_R^T, W_O^T) is a full column-rank matrix.

Because target variables are meant as "information adding", these conditions are usually attained without any need to tweak the original equation set.

In the case of our example of modular computation, for example, all target variables are bits (i.e., have values in $\{0, 1\}$). In this common situation, each target variable is an L_i, and the definition of L_i as a bit involves the following equations:

$$L_i - R_i = 1, \tag{3}$$

$$O_i = 0. \tag{4}$$

The reason this defines L_i as a $\{0, 1\}$-variable is that coupled with the standard requirement (1), this equation set indicates that the value $x = L_i$ satisfies $x(x - 1) = 0$.

Importantly, however, regardless of how this bit is later used, it already satisfies all three of our assumptions regarding target variables:

(1) The target variable is L_i, not O_i.
(2) The corresponding R_i is not a target variable. (Indeed, it adds no new information to the system, given L_i.)
(3) Equation (3) contains only the L_i coefficient as a nonzero coefficient in (W_L^T, W_R^T, W_O^T), and Equation (4) contains only the O_i coefficient. As such, neither the L_i column nor the O_i column can be represented as a linear combination of the other columns in the matrix.

If any O_i does happen to be a target variable, it is always possible to represent it as an L or R variable for our purposes, e.g. by creating a new L_j variable and introducing a new equation,

$$L_j = O_i,$$

and similar means can resolve a situation where both L_i and R_i are required to be target variables. However, we expect the need for such additional equations to be very rare.

The rest of this paper will prove the correctness of Algorithm 1.

5 PROOFS OF CORRECTNESS

The correctness of Algorithm 1 is described by the following three theorems.

Theorem 1. *If the Prover knows a solution L' and R' for the target variables (with the previously-stated external commitments) such that for any choice of r the Prover knows a solution L'' and R'' for the proof variables, for which these variables, jointly, form a solution to the equation set described by $(W_L', W_L'', W_R', W_R'', W_O', W_O'', W_v, c)$ that is generated by r, then the Prover can successfully complete Algorithm 1.*

Theorem 2. *If the Prover does not know a solution L' and R' for the target variables (with the previously-stated external commitments) such that for a random choice of r the Prover knows, with non-negligible probability, a solution L'' and R'' for the proof variables, such that these variables, jointly, form a solution to the equation set described by $(W_L', W_L'', W_R', W_R'', W_O', W_O'', W_v, c)$ that is generated by r, then the Prover cannot, other than with negligible probability, successfully complete Algorithm 1.*

Theorem 3. *Algorithm 1 is a zero-knowledge algorithm.*

The conjunction of Theorem 1 and Theorem 2 ensures that a Verifier witnessing a Prover successfully complete Algorithm 1 can confidently view this as a successful proof (e.g., of knowledge) by

Algorithm 1 Computing the internal commitments.

```
1: Let $\{L'_i\}_{i=1}^m$ be such that $L'_i = L_i$ if $L_i$ is a target variable and zero otherwise.
2: Let $L'' = L - L'$.
3: Let $R'$ and $R''$ be defined analogously, but for variables in $R$.
4: The Prover commits to $A'_I = \langle L' \cdot G\rangle + \langle R' \cdot H\rangle + \tilde{a}'\tilde{B}$.
5:                                        ▷ $\tilde{a}'$ is a random blinding variable.
6: The Prover commits to $S = \langle s^L \cdot G\rangle + \langle s^R \cdot H\rangle + \tilde{s}\tilde{B}$.
7: ▷ $s^L$, $s^R$ and $\tilde{s}$ are all random blinding variables as in standard Bulletproofs.
8: The Verifier supplies a random $r \in \mathcal{F}$.
9: Using $r$, the Prover and Verifier both independently compute the randomly-chosen equation set $(W'_L, W''_L, W'_R, W''_R, W'_O, W''_O, W_v, c)$. Here, $W_L = W'_L + W''_L$, $W_R = W'_R + W''_R$, and $W_O = W'_O + W''_O$, but $W'_L$ and $W'_R$ are zeros except at target variables, and $W''_L$ and $W''_R$ are zeros except at proof variables. The matrices $W'_O$ and $W''_O$ are defined analogously, where $W'_O$ is zero, except where either $L$ or $R$ are target variables.
10: The Prover commits to $A''_I = \langle L'' \cdot G\rangle + \langle R'' \cdot H\rangle + \tilde{a}''\tilde{B}$.
11:                                        ▷ $\tilde{a}''$ is a random blinding variable.
12: The Verifier supplies a random value $q \in \mathcal{F}$.
13: Let $O' = (qL' + L'') \cdot (qR' + R'')$.
14: The Prover commits to $A_O = \langle O' \cdot G\rangle + \tilde{o}\tilde{B}$.
15:                                        ▷ $\tilde{o}$ is a random blinding variable.
16: The Prover continues with the standard Bulletproof protocol, but using an equation set with $W_L = q^{-1}W'_L + W''_L$, $W_R = q^{-1}W'_R + W''_R$, $W_O = q^{-1}W'_O + W''_O$, and the previously-computed $W_v$ and $c$.
17: The Verifier continues with the standard Bulletproof protocol, but using $A_I = qA'_I + A''_I$, together with $A_O$ and $S$, as the Prover's internal commitments.
18:                    ▷ One known solution to this equation set that should match the commitments is $\bar{L} = qL' + L''$, $\bar{R} = qR' + R''$, $\bar{O} = O'$. The Prover can use this throughout the remainder of the proof, with the synthetic blinding factor $\tilde{a} = q\tilde{a}' + \tilde{a}''$.
```

the Prover, while Theorem 3 ensures that by running the protocol of Algorithm 1 the Verifier receives no more information than in standard Bulletproofs. Together, this is what we wanted to ascertain.

We will now prove these theorems in turn.

Proof of Theorem 1. As mentioned in Algorithm 1, the Prover can continue the Bulletproof as usual, using $\bar{L} = qL' + L''$, $\bar{R} = qR' + R''$ and $\bar{O} = O'$ as the (secret) solution to the equation set, and using $\tilde{a} = q\tilde{a}' + \tilde{a}''$ as a synthetic blinding variable, in place of the standard $\tilde{a}$ blinding variable used for the commitment A_I in normal Bulletproofs.

To see that this works, we need to prove two things:

(1) That this solution does solve the new equation set, and
(2) That this solution matches the commitments A_I, A_O and S.

The solution described by $\bar{L}$, $\bar{R}$ and $\bar{O}$ retains the original values of all proof variables and external variables, but multiplies the values of the internal target variables in L and R by the factor q.

The value of $\bar{O}$ is computed at line 13 of Algorithm 1 already after the value of q is known. Note that while L_i and R_i are multiplied by q if they are internal target variables, $O_i = L_iR_i$ is multiplied by q if either L_i or R_i is an internal target variable. (By assumption, they cannot both be target variables, so no q^2 factor is ever introduced.) Thus, each O_i is multiplied by the appropriate factor to satisfy every nonlinear equation in the R1CS equation set.

This leaves only the linear equations. However, because each element in the solution has been multiplied by a known factor (either 1 or q), by simply dividing each coefficient by the same factor, the total contribution of each variable to each equation remains the same, and so all equations continue to balance.

This manipulation of the coefficients is exactly what defines the coefficients (W_L, W_R, W_O, W_v, c) of the new equation set, on line 16 of Algorithm 1. Hence, the entire R1CS equation set is satisfied.

As for the commitments: first, the commitment on A_O occurs after knowing q, so can certainly be made to fit, and second, the commitment on S is independent of q and r, so, again, is not problematic.

This leaves only the commitment on A_I, which is synthetic, and computed as $A_I = qA'_I + A''_I$ on line 17 of Algorithm 1. However, it is straightforward to see that this commitment matches the proposed solution: by construction

$$A'_I = \langle L' \cdot G\rangle + \langle R' \cdot H\rangle + \tilde{a}'\tilde{B},$$

and

$$A''_I = \langle L'' \cdot G\rangle + \langle R'' \cdot H\rangle + \tilde{a}''\tilde{B},$$

so

$$\begin{aligned} A_I &= qA'_I + A''_I \\ &= \langle (qL' + L'') \cdot G\rangle + \langle (qR' + R'') \cdot H\rangle + (q\tilde{a}' + \tilde{a}'')\tilde{B} \\ &= \langle \bar{L} \cdot G\rangle + \langle \bar{R} \cdot H\rangle + \tilde{a}\tilde{B}, \end{aligned}$$

where $\tilde{a}$ is, as described above, a synthetic blinding factor whose value can easily be determined by the Prover through the computation $\tilde{a} = q\tilde{a}' + \tilde{a}''$. □

Proof of Theorem 2. In the standard Bulletproof, the Prover begins by committing to the external variables, followed by commitments A_I, A_O and S. These are commitments made before any input by the Verifier, so the Prover has complete freedom in deciding how to compute them. However, the mechanics of the Bulletproof ensure that if the Prover successfully completed a Bulletproof then this proves, except with negligible error probability, that the Prover's initial commitments correspond to (L, R, O) values, known by the Prover, that satisfy the desired equation set. Thus, the Bulletproof itself ensures that the Prover knows a solution to the stochastically-generated equation set.

The only element of Theorem 2 that is not guaranteed by the Bulletproof itself is that the Prover must use in all such solutions, regardless of the Verifier's choice of r, the same assignments L', R' to the target variables (or the proof has only negligible probability to complete successfully). We will prove that this is guaranteed by Algorithm 1.

Let us begin by considering the synthetic commitment

$$A_I = qA'_I + A''_I.$$

The Bulletproof assures us that the Prover is able to describe A_I as a linear combination of $\{G_i\}_{i=1}^m$, $\{H_i\}_{i=1}^m$ and $\tilde{B}$. The fact that the Prover cannot do the same to a random element of $\mathcal{G}$ is the core cryptographic assumption underlying the strength of Bulletproofs.

From this we can conclude that the Prover can also describe A'_I and A''_I in the same way: if the Prover is able to describe A'_I as such a linear combination, then the description of A''_I can be retrieved simply from $A_I - qA'_I$; if, on the other hand, the Prover is unable to describe A'_I in this way, then qA'_I (and by extension also $qA'_I + A''_I$) is effectively a random element of $\mathcal{G}$, in which case the Prover would not have been able to describe A_I as a linear combination, contrary to the assumption.

The Prover is therefore guaranteed (except with negligible probability) to know solutions L', L'', R', R'', $\tilde{a}'$, $\tilde{a}''$ to

$$A'_I = \langle L' \cdot G\rangle + \langle R' \cdot H\rangle + \tilde{a}'\tilde{B},$$
$$A''_I = \langle L'' \cdot G\rangle + \langle R'' \cdot H\rangle + \tilde{a}''\tilde{B}.$$

What we need to prove is that the coefficients of L'' and R'' must necessarily all be zeroes in their respective target variables. When this is the case, the solutions $\bar{L} = qL' + L''$ and $\bar{R} = qR' + R''$, in their target variable components, are invariant to L'' and R'', which is what we want to prove.

To see that this is the case, consider the equation set ultimately used in the Bulletproof. This can be described by the parameters (W_L, W_R, W_O, W_v, c), which were computed from their constituents $(W'_L, W''_L, W'_R, W''_R, W'_O, W''_O, W_v, c)$.

This ultimate equation set takes the form

$$\begin{aligned}(q^{-1}W'_L + W''_L)(qL' + L'') + (q^{-1}W'_R + W''_R)(qR' + R'')\\ +(q^{-1}W'_O + W''_O)\left((qL' + L'') \cdot (qR' + R'')\right) = W_v v + \vec{c}.\end{aligned}$$

Because the right-hand side is independent of q, we can conclude that the left-hand side is, too. This allows us to split the resulting left-hand side equation to multiple terms, each a coefficient multiplying a different power of q, and to determine regarding all such terms that have a dependence on q that they must equal zero. Specifically, if we consider only the terms multiplied by a q^{-1} factor, we get

$$W'_L L'' + W'_R R'' + W'_O \left(L'' \cdot R''\right) = 0.$$

This equation describes a linear combination of the columns of (W'_L, W'_R, W'_O) that equals zero. By construction, the columns of (W'_L, W'_R, W'_O) are all zeros, except at internal target variables. Reducing these to only the nonzero columns, we get the matrix (W^T_L, W^T_R, W^T_O) regarding which we assumed that it has full column rank. Thus, any linear combination of these columns yielding zero must be the trivial one, and consequently both L'' and R'' must be zero at all positions corresponding to internal target variables.

In other words, the portion of the commitment A''_I that relates to target variables is all zeroes, and the Prover has not changed their choice of target variables between committing to them with A'_I (before knowing r) and making the final synthetic commitment A_I (after knowing r). □

Proof of Theorem 3. The information sent by the Prover to the Verifier when using Algorithm 1 is the same as when using standard Bulletproofs, with the exception that whereas in Bulletproofs the Verifier receives the commitment A_I, in Algorithm 1 the Verifier receives A'_I and A''_I, and computes on their own $A_I = qA'_I + A''_I$ as a synthetic commitment.

Given that there is a known linear equation connecting A'_I, A''_I and A_I, we can say that the information received by the Verifier (i.e., A'_I and A''_I) is informationally-equivalent to receiving A'_I and A_I.

This shows that the information content received by the Verifier when using Algorithm 1 is exactly the same as when using standard Bulletproofs, plus the addition of any information contained in A'_I.

In fact, a Prover with prior knowledge of q and r would have been able to compute A_I first (as in standard Bulletproofs), and to derive A'_I and A''_I from it, later. Given that the Prover knows a legitimate (L', L'', R', R'') solution to the equation set, all that is needed for this is to choose blinding factors that satisfy the equation

$$\tilde{a} = q\tilde{a}' + \tilde{a}''.$$

To satisfy this, however, both $\tilde{a}$ and $\tilde{a}'$ can be chosen randomly, uniformly and independently, with $\tilde{a}''$ computed later as $\tilde{a}'' = \tilde{a} - q\tilde{a}'$.

In total, we therefore know the following. First, the information content delivered to the Verifier is the same as in a standard Bulletproof, plus whatever information is contained in A'_I. Second, $\tilde{a}'$ can be chosen randomly, uniformly and independently of any part of the standard Bulletproof, and consequently, due to the blinding factor $\tilde{a}'\tilde{B}$, the value of A'_I is both uniformly distributed in $\mathcal{G}$ and independent of all other information sent to the Verifier as part of the Bulletproof.

We conclude that the commitments A'_I and A''_I convey no new information to the Verifier, and the proof remains zero-knowledge. □

6 CONCLUSIONS

The ability to construct one's proof of knowledge as a stochastic, interactive protocol is at the heart of the entire field of interactive proofs, and zero-knowledge proofs in particular, and has been used to create short and efficiently-checkable proofs for proving the knowledge of a solution to large deterministic equation sets.

In this paper, we extend this approach further, by demonstrating that knowledge can sometimes be encoded more efficiently as a solution to a stochastically-generated equation set, rather than a deterministic one.

We introduce a protocol extending the standard Bulletproof protocol, which allows such stochastic equation sets to be represented, and knowledge of their solutions proved, while maintaining all the classical benefits of the Bulletproof approach: efficient proof generation, efficient proof checking, extremely short proofs, and the ability to use Fiat-Shamir challenges in order to turn an interactive proof into a non-interactive proof.

Using stochastic equation sets, we can reduce the number of variables required for a proof by orders of magnitude, leading to a corresponding order-of-magnitude saving in both proving times and proof checking times, as well as shorter proofs.

ACKNOWLEDGMENTS

This work was funded by INCERT GIE.

REFERENCES

[1] Leonidas Alaoglu and Paul Erdős. 1944. On highly composite and similar numbers. *Trans. Amer. Math. Soc.* 56 (1944), 448–469.
[2] Sanjeev Arora and Boaz Barak. 2009. *Computational complexity: A modern approach.* Cambridge University Press.
[3] Sanjeev Arora and Shmuel Safra. 1998. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM (JACM)* 45, 1 (1998), 70–122.
[4] László Babai. 1985. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing.* 421–429.

[5] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. 1990. Everything provable is provable in zero-knowledge. In *Advances in Cryptology—CRYPTO'88: Proceedings 8*. Springer, 37–56.
[6] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. 2019. Multi-prover interactive proofs: How to remove intractability assumptions. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 373–410.
[7] Daniel J Bernstein. 2006. Curve25519: New Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography*. Springer, 207–228.
[8] Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. 2008. Twisted Edwards curves. In *International Conference on Cryptology in Africa*. Springer, 389–405.
[9] Daniel J Bernstein and Frank Denis. 2019. Libsodium – A modern, portable, easy to use crypto library. https://github.com/jedisct1/libsodium. Retrieved 3 February 2023.
[10] Gilles Brassard, David Chaum, and Claude Crépeau. 1988. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences* 37, 2 (1988), 156–189.
[11] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 315–334.
[12] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms* (third ed.). MIT press, Cambridge, Chapter 34.2: Polynomial-time verification, 1061–1066.
[13] Ivan Damgård, Torben Pedersen, and Birgit Pfitzmann. 1994. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In *Advances in Cryptology – CRYPTO'93*. Springer, 250–265.
[14] Morris J Dworkin. 2015. *SHA-3 standard: Permutation-based hash and extendable-output functions*. Technical Report Federal Information Processing Standards Publication (FIPS PUBS) 202. National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C.
[15] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Crypto*, Vol. 86. Springer, 186–194.
[16] Oded Goldreich. 2001. *The Foundations of Cryptography*. Vol. 1. Cambridge University Press, Cambridge, Chapter 4: Zero-Knowledge Proof Systems, 184–330.
[17] Mike Hamburg. 2015. Decaf: Eliminating cofactors through point compression. In *Advances in Cryptology – CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16–20, 2015, Proceedings, Part I 35*. Springer, 705–723.
[18] David Naccache and Igor E Shparlinski. 2009. Divisibility, smoothness and cryptographic applications. In *Algebraic Aspects of Digital Communications*, Tanush Shaska and Engjell Hasimaj (Eds.). Vol. 24. Ios Press.
[19] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*. Springer, 129–140.
[20] Dingyi Pei, Arto Salomaa, and Cunsheng Ding. 1996. *Chinese remainder theorem: Applications in computing, coding, cryptography*. World Scientific.
[21] Srinivasa Ramanujan. 1997. Highly composite numbers. *Ramanujan Journal* 1 (1997), 119–119.
[22] Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* 8, 2 (1960), 300–304.
[23] Peter Schwabe and Amber Sprenkels. 2019. The complete cost of cofactor $h = 1$. In *Progress in Cryptology – INDOCRYPT 2019: 20th International Conference on Cryptology in India, Hyderabad, India, December 15–18, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11898)*, Feng Hao, Sushmita Ruj, and Sourav Sen Gupta (Eds.). Springer, 375–397.
[24] Harold N Shapiro. 2008. *Introduction to the Theory of Numbers*. Courier Corporation.
[25] Michael Sipser. 2013. *Introduction to the Theory of Computation* (third ed.). Cengage Learning, Boston, MA, Chapter 10.4: Interactive Proof Systems, 415–427.
[26] John Viega, Matt Messier, and Pravir Chandra. 2002. *Network Security With OpenSSL: Cryptography for Secure Communications*. O'Reilly Media, Inc.
[27] David Wong. 2021. *Real-world cryptography*. Simon and Schuster.
[28] Andrew Chi-Chih Yao. 1979. Some complexity questions related to distributive computing. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*. 209–213.
[29] Eric A Young, Tim J Hudson, and Ralf S Engelschall. 2022. OpenSSL. https://www.openssl.org/.

A TECHNICAL DETAILS OF THE EXAMPLE EQUATION SET

The following example is taken from a proof-of-concept for a digital wallet app that we designed and implemented, and then optimised using the stochastic equation set approach, for INCERT GIE, the Luxembourgian public agency in charge of, among other things, securing digital signatures in national identity documents.

The description omits many aspects of the project that are not relevant for the present discussion, but describes the underlying equation set for the Bulletproof used, in both deterministic and stochastic versions, exactly, without any simplifications.

A.1 The problem scenario

We want to design a digital wallet app, meant for the storage of official documents. The design of the app should enable wallet owners to certify facts about themselves from official documents, including in unofficial scenarios.

For example, a wallet owner may want to use the wallet in order to certify to a bartender that they are above the legal drinking age.

However, because the bartender is not trusted, the wallet owner wants to make this certification in zero knowledge. In particular, no trackable identifier of the wallet owner may be given to the bartender.

In this scenario, the wallet owner needs to show

(1) That they possess a government certificate stating that Person X is above the legal drinking age, and
(2) That they are Person X,

without divulging any personally identifying information, or any trackable identifier of any kind.

As part of the problem setup, it is assumed that such verifications are done "offline", in the sense that neither party can be assumed to have a network connection. The entire verification process includes only the Prover and the Verifier, without any third party involvement.

The solution we opted for is as follows.

First, we assign to each wallet owner a private identifier and a corresponding public identifier. The public identifier is only "public" in the sense that it can be divulged to official entities; it cannot be divulged to non-trusted entities such as the bartender. For this, each wallet owner allots two random prime numbers of length 1024 bits, u_p and u_q. The numbers u_p and u_q serve as the wallet owner's private identifier, whereas the product $u_n = u_p u_q$ is their public identifier.

Second, any document in the wallet is signed by the issuing government authority, to certify its veracity. Let n be a 4096-bit RSA public key, publicly known and associated with the issuing government authority. The issuing authority signs documents, M, by means of the signature $S = M^d \bmod n$, where d is chosen such that $M = S^3 \bmod n$.

The message M is constructed as the concatenation of u_n (the public key identifying the wallet owner), I (the document information, such as an attestation that the wallet owner is above the legal drinking age), and A (additional random padding, completing to 4096 bits). In our example, we use 132 bytes for I, leaving an additional 992 bits for A.

In integer arithmetic, the equation that the wallet owner needs to prove they know a solution for is

$$S^3 = u_p u_q + I \times 2^{2048} + A \times 2^{3104} + Dn, \tag{5}$$

where S, u_p, u_q, A and D are all private integer variables, known only to the wallet owner, whereas I is communicated to the Verifier, and n is public knowledge. Regarding the private integers, only their bit size is known. In the case of u_p and u_q, we can also pre-set their top and bottom bits to 1.

As long as the user can prove the bit lengths of all private variables, this equation encapsulates both that the issuing authority signed the document with the appropriate public identifier $u_n = u_p u_q$, and that the wallet owner knows the corresponding private identifier (u_p, u_q), by this proving that the wallet owner really is the object of the document.

A.2 Standard Bulletproof formulation

To meet the requirements of the problem scenario, we implemented the Bulletproof (initially in its standard formulation) over the additive Curve25519 [7] twisted Edwards elliptic curve group [8]. We used Ristretto [23], an extension of Decaf [17], in order to eliminate cofactors. Cofactor elimination is necessary in order to reduce the group to a prime order, as required for Bulletproofs.

To reformulate the integer equation (5) as an equation modulo the group size, note that the entire equation is of $4096 \times 3 = 12288$ bits, as determined by its largest element, S^3. Thus, it is enough to ascertain that the difference between the left-hand side and the right-hand side is zero modulo some large value N, greater than 2^{12288}.

To prove this using the tools of standard Bulletproofs, we add to the shared public data (e.g., data published by the certificate-issuing authority) the identity of 174 distinct primes, each 71 bits in length, $P_1 \ldots, P_{174}$, such that $N = \prod_i P_i$ is sufficiently large. The computation of (5) can then be performed modulo each P_i separately, and if the equation holds under all moduli, then by the Chinese Remainder Theorem, it must also hold modulo N.

Using 71-bit-long primes allows us to compute the modular equivalent of (5) without overflows.

To ensure that all variables are of the appropriate bit length, we introduce separate Bulletproof variables for each bit of S, u_p, u_q, A and D. For example, let us define L_S^j to be the j'th bit of S. Then the following equations

$$R_S^j = L_S^j - 1,$$
$$L_S^j \times R_S^j = 0,$$

ensure that all L_S^j are bits. Defining S (not a Bulletproof variable) as

$$S = \sum_{j=0}^{4095} L_S^j 2^j$$

then ensures that it is of the appropriate bit length.

In our case, we modify this paradigm by computing a new Bulletproof variable, S_i, as

$$S_i = \sum_{j=0}^{4095} L_S^j (2^j \bmod P_i), \tag{6}$$

to act as a convenient stand-in for $S \bmod P_i$, and we create in the same way also modular variants for the equation's other variables.

This modular representation works because

$$S_i \equiv S \pmod{P_i}.$$

However, unlike "$S \bmod P_i$", the value of S_i is not constrained to be in $[0, P_i)$. Instead, we can only be sure that it does not exceed $4096P_i$.

This is enough to ensure that S_i^3 is still within the 252 bit range of the finite field in which we are computing, allowing us to effectively compute all of (5) modulo each P_i, without overflows.

If we refer to the modular version of the left-hand side of (5) as LHS_i and the right-hand side as RHS_i, then the final R1CS equation for each P_i becomes

$$LHS_i + C_i = RHS_i + div_i P_i. \tag{7}$$

The constants C_i, known to both parties, are multiples of P_i designed to make sure that the div_i values proving the modular equation are nonnegative.

In total, this approach requires 4096 variables in order to store the bits of S, twice 1022 bits in order to store the bits of u_p and u_q (recalling that their most and least significant bits are both known to be 1), a further 992 variables to store the bits of A, 8192 variables to store the bits of D, and 178 bits for each of the 174 div_i variables. Each such bit requires one multiplication gate.

Additionally, for each of the 174 P_i values, 2 multiplication gates are required to compute S_i^3 and 1 for the modular equivalent of the multiplication operation $u_p \times u_q$.

In total, these are $46,818$ multiplication gates.

Bulletproofs, by their design, require a number of multiplication gates that is a power of 2, for which reason this number must be rounded up to $65,536$.

The final $18,719 = 65,536 - 46,818$ multiplication gates added relate to variables that do not participate in any equation. Though technically, in the terminology introduced in this paper, such variables are "proof variables", from a stand-point of performance comparison it makes more sense to separate them to their own category. We refer to them here as *dummy variables*.

A.3 A stochastic Bulletproof design

The time costs of our standard Bulletproof implementation can largely be traced back to two sources. First, setting up the Bulletproof equations requires much large-integer arithmetic, which we implemented using the BIGNUM functionality of OpenSSL [26, 29]. The bulk of these computations need to be repeated 174 times, one for each prime modulus.

Second, the Bulletproof itself requires a one-time amount of computation related to the total number of nonzero coefficients in the Bulletproof equations, followed by a sequence of computations whose complexity is linear in the number of multiplication gates. These computations are in both Curve25519 Ristretto and in the modular integer field of the same size, and were implemented using the Ristretto implementation of the libsodium library [9]. Note, however, that of our Bulletproof variables only $15,324 = 4096 + 1022 \times 2 + 992 + 8192$ are target variables. Each of the 174 moduli tested introduces 181 new multiplication gates, all composed solely of proof variables. In total, this adds up to $181 \times 174 = 31,494$ multiplication gates for the proof variables, making these the majority of multiplication gates required.

In both cases, our standard implementation is weighted down by the fact that we need to check all 174 moduli. Using a stochastic Bulletproof, we can reduce this number by selecting moduli randomly, from a larger pool.

To do this, we must first revert to a "less optimised" version of (5). In (5), we minimised the total number of multiplication gates used by working in arithmetic modulo 71-bit primes. This allowed us to compute S_i^3 without overflows.

For the stochastic design, because we need our moduli to be selected from a larger pool, we require our computation to be modulo higher numbers, so instead of encapsulating (5) by a single equation, we break it down to

$$S' = S^2,$$
$$S'S = u_p u_q + I \times 2^{2048} + A \times 2^{3104} + Dn,$$

where these, in turn, translate to the modular equations

$$S_i^2 + \mathit{offset}_1 = S_i' + \mathit{div}_i' P_i$$
$$S_i' S_i + \mathit{offset}_2^i = u_n^i + A^i + D^i + M_i P_i,$$
$$u_n^i = u_p^i u_q^i,$$

where A^i and D^i are the modular representations of $A \times 2^{3104}$ and Dn, respectively, and offset_2^i, beyond being a modular offset, also encapsulates $(I \times 2^{2048}) \bmod P_i$. Each of the three equations above contributes one multiplication gate per modulus.

In the new formulation, we use the 252-bit-sized ring only for squaring, not for cubing numbers. By setting each P_i to be at most 111 bits in length, we can ensure that all S_i and other modular representations of our various integer variables all stay within 123 bits, making products of two such variables limited to 246 bits. This affords enough headroom to ensure that all of (5) can be computed in less than the available 252 bits.

We call this a "less optimised" version of the base equation, because in the deterministic formulation testing the equation's correctness modulo any P_i required us to introduce only 181 new multiplication gates, and in this new formulation a full 388 are required: 111 for the bits of S_i', 136 for the bits of div_i', 138 for the bits of M_i, and one for each of the 3 modular equations.

Consider, therefore, verifying (5) modulo some randomly-chosen 111-bit number, Q, not necessarily prime.

We do not wish to choose Q uniformly in this space, because if the difference, Δ, between the right-hand side and the left-hand side of (5) is a highly composite number [1, 21], the number of Q values dividing Δ will be large, making the modular equation correct under many potential moduli, even though the integer equation is incorrect.

Instead, we choose Q uniformly from those numbers that are 2200-rough [18], defined as those numbers all of whose divisors are greater than 2200.

If Δ, the discrepancy in the integer equation, is nonzero and at most 12288 bits long, it can have at most 1106 prime divisors over 2200.

For a 111-bit number to be 2200-rough, on the other hand, it must be the product of at most 9 primes over 2200. Note that if some subset of the prime divisors over 2200 of Δ has a product that is 111 bits in length, no product of any strict subset of this set can be in the same range (as these products must differ by at least 11 bits of length). Thus, the maximum number of divisors of Δ that are 2200-rough integers 111 bits in length can be bounded at C_9^{1106}, i.e. the number of ways to choose 9 primes out of 1106. (This can be attained in the worst case, which is when Δ is both 2200-rough and square free [24], meaning that it factorises into only unique primes.)

This number of possibilities is just under $2^{72.5}$.

The total number of 2200-rough 111-bit numbers is well-approximated as $2^{110} \prod_{i:p_i \le 2200} \frac{p_i - 1}{p_i}$, where p_i is the i'th smallest prime. This is approximately 0.073×2^{110}.

In total, the chance for a randomly-chosen rough Q to divide a non-zero Δ can be bounded from above by $2^{-33.7}$. By choosing 2 such Q candidates at random, we ensure that the probability for false acceptance is bounded from above by $2^{-67.4}$, which was deemed an acceptable rate.

In the stochastic version, only 2 moduli need to be checked, rather than 174, reducing the bulk of the big-number computations required for the algorithm.

In total, in addition to the 15,324 multiplication gates used in the proof for target variables, our stochastic implementation allows us to reduce the number of multiplication gates dedicated to proof variables from $181 \times 174 = 31,494$ to $388 \times 2 = 776$, a reduction by a factor of 41. This reduces the total number of multiplication gates to under 2^{14}, a saving by a factor of 4 overall.

A.4 Performance comparison

We implemented both algorithms in C, utilising OpenSSL's BIGNUM library and libsodium Ristretto, as described above. For the stochastic version, we used the algorithm described in Appendix B in order to allot uniform rough moduli.

Table 1 summarises the performance comparison between the standard Bulletproof implementation and the one using a stochastic equation set. Timing results in the table are given based on 10 runs. These runs were executed on the following architectures. Note that while we provide full details for each architecture, our Bulletproof code was, in fact, single-threaded code. It also did not take advantage of the GPU that was available in each architecture.

1. A Lenovo Legion 5 laptop, with an i7-12700H processor (6 P-Cores at 2.30GHz, 8 E-Cores at 1.7GHz), and 32GB of RAM,
2. A Samsung Galaxy S22+ (Octa-core, with one 2.995GHz Cortex-X2 processor, three 2.496GHz Cortex-A710, and four 1.78GHZ Cortex-A510, and 8GB of RAM),
3. A Samsung Galaxy A20 (Octa-core, with two 1.6GHz ARM Cortex-A73 processors and six 1.35GHz ARM Cortex-A53 ones, and 3GB of RAM).

The latter two are Android devices, for which the Bulletproof was implemented as part of an Android mobile app.

In the table, "non-target variables" relate to both proof variables and dummy variables, while "proof variables" excludes dummy variables.

The table shows that the stochastic equation set approach outperforms the standard approach in every metric, usually by substantial margins, without compromising any of the features that make Bulletproofs attractive in the first place.

Even though the Galaxy A20 is a low-end device, running the Bulletproof at only 10% the speed of the laptop, and the Galaxy S22+ is a high-end device, reaching 2/3 of the speed of the laptop, all three architectures show the same consistent improvement of roughly 4X the speed between the deterministic and stochastic Bulletproof implementations.

Table 1: Performance comparison

		Deterministic	**Stochastic**	**Improvement**
	Multiplication gates	65, 536	**16, 384**	4X
	(of which for non-target variables)	50, 212	**1, 060**	47.4X
	(of which for proof variables)	31, 494	**776**	40.6X
	Linear equations	94, 854	**32, 204**	2.9X
	Proof size (bytes)	1, 440	**1, 344**	96
Lenovo Legion	Proof generation time [$\mu \pm 2\sigma$] (s)	45.8 ± 1.2	**10.8 ± 0.5**	4.2X
	Verification time [$\mu \pm 2\sigma$] (s)	18.2 ± 0.6	**4.3 ± 0.2**	4.3X
Samsung Galaxy S22+	Proof generation time [$\mu \pm 2\sigma$] (s)	66.3 ± 1.3	**16.6 ± 0.8**	4.0X
	Verification time [$\mu \pm 2\sigma$] (s)	25.2 ± 1.1	**6.3 ± 0.8**	4.0X
Samsung Galaxy A20	Proof generation time [$\mu \pm 2\sigma$] (s)	440.4 ± 1.6	**108.0 ± 1.0**	4.1X
	Verification time [$\mu \pm 2\sigma$] (s)	171.0 ± 1.8	**40.4 ± 1.3**	4.2X

B CHOOSING A ROUGH MODULUS

In this section, we discuss the question of how to efficiently allot a uniform 2200-rough number that is 111 bits long, recalling that the procedure to do so must be deterministic from one or more Fiat-Shamir challenges (so that it can be repeated by both Prover and Verifier, ascertaining that the number was not chosen adversarially). The distribution resulting from our algorithm will not be completely uniform, but will have high enough entropy for our purposes.

Our method employs two techniques.

First, in order to allot a number that is not divisible by 2, 3, 5, 7 and 11, we prepare in advance a table, T, of all 480 residues modulo $2310 = 2 \times 3 \times 5 \times 7 \times 11$ that do not divide by any of these primes.

In our implementation, we used Fiat-Shamir challenges that are outputs of SHA3-512 [14]. This provides us, for a single challenge, with 512 pseudorandom bits. Of these, we can use 17 bits in order to compute the 480 modulus of a uniformly-chosen 17-bit number, and this already provides us with a close-enough approximation of a uniform value in the range $[0, 479]$, which we can use as an index to the look-up table, T, of residues modulo 2310, thus uniformly choosing a residue class for our chosen number, Q, modulo 2, 3, 5, 7 and 11. Let m_{2310} be the value retrieved from the table of residues.

Next, we choose uniformly a value d_{2310} in the range

$$[\lceil 2^{110}/2310 \rceil, \lfloor 2^{111}/2310 \rfloor).$$

This provides us with a Q candidate $2310 \times d_{2310} + m_{2310}$ which we know does not divide by any prime smaller than 13.

The range for d_{2310} is just under 99 bits. We allot a d_{2310} value uniformly by choosing 99 bits from the SHA3 output, interpreting it as a random number, r, of up to 99 bits in length, and computing d_{2310} as $r + \lceil 2^{110}/2310 \rceil$. We then use rejection sampling to reject the choice if the value of d_{2310} this produces is not smaller than $\lfloor 2^{111}/2310 \rfloor$. Such rejection happens with probability around $1/8$.

Next, we need to make sure that Q does not divide by any of the 322 remaining primes below 2200. This we do by checking, i.e. by more rejection sampling: if our Q candidate divides by any of the remaining "small" primes, we eliminate it, and use the next 99 bits of the SHA3 output in order to allot another d_{2310},

The probability that a Q candidate does not divide any of the remaining 322 primes is approximately $1/3$, and even when accounting also for the probability of rejection due to a too-large d_{2310}, the probability of success is still above 30%. In expectation, roughly 3 attempts are needed in order to find a successful candidate. Having started with 512 random bits, we can afford a full 5 attempts before another Fiat-Shamir challenge needs to be computed, so in high probability the procedure can be completed using only the first challenge.

This leaves us with the question of how to efficiently verify that a candidate value Q does not divide by any of the remaining primes. To do this, we have partitioned the primes to 49 buckets, such that the primes in each bucket have a product smaller than 2^{64}. This can be done simply by greedy assignment.

The trick here is that much of the reason for the slow computation time is the need to handle large numbers (which we do using OpenSSL's BIGNUM functionality). Instead, for each Q candidate we only compute in this way the 49 residues $\{Q \bmod \Pi_i\}_{i=1}^{49}$, where Π_i is the product of all primes in the i'th bucket, which is a value that we pre-compute. The result of each such modulo operation can then be stored in a 64-bit unsigned "long" integer, allowing all remaining checks to be done at much higher speeds.