

Argo: Leaking a Secret from an Invalid Groth16 Proof

Liam Eagen Ying Tong Lai

Glass Coins

Abstract. We present Argo, a randomized encoding scheme for the invalidity of pairing-based SNARKs like Groth16. Argo allows an encoder to encode a proof such that a decoder can derive a secret if the proof is invalid. We prove the selective security of invalidity Argo, as well as the adaptive security of validity Argo, under DDH in one source group of a bilinear pairing in the standard model, a standard assumption that holds in the generic group model used to analyze Groth16.

We show how to couple invalidity Argo with a projective garbling scheme like Argo MAC or Duty-Free Bits to construct a garbled circuit for Groth16 invalidity as a function of the proof bits, and how to use it with BitVM to construct a permissionless, optimistic Groth16 verifier for Bitcoin: a decoder proves a proof invalid by publishing a secret, and the decoder can be anyone, not necessarily known in advance.

When instantiated with BN254, Argo has an offline encoding size of about 21 kB, a thousand times smaller than the projective garbling scheme, and an end-to-end protocol size more than three orders of magnitude smaller than existing Yao-based approaches. Generating an Argo + Argo MAC garbled circuit takes tens of milliseconds on a laptop, compared to minutes for a Yao GC. Argo is the first protocol that makes permissionless, optimistic SNARK verification on Bitcoin practical.

1 Introduction

SNARK verification enables blockchains like Ethereum [49] to verify complex off-chain statements efficiently in the protocol. They are used to build complex systems like zk-Rollups [46] and add privacy to payments. However, Bitcoin, the first and largest Blockchain by market cap, cannot verify SNARKs. Its scripting language has no opcodes for field arithmetic or pairings, and the transaction size needed to emulate them is far beyond what a block can hold. Naturally, this leads to the question: if we cannot verify a SNARK on Bitcoin, how close can we get?

The BitVM [42,43,48] line of work is one answer to this question. Rather than directly verifying a SNARK, we can build an optimistic SNARK verifier. In such a protocol, a prover submits a claimed proof on chain, and a verifier is able to submit an efficient fraud proof that the SNARK is invalid. There have been many variants of the BitVM protocol, and the most recent versions are descended from Delbrag [44] which uses a garbled circuit as this fraud proof

mechanism. The prover commits to the proof on-chain by revealing input labels, and the verifier derives a secret using the garbled circuit if the proof is invalid. All Bitcoin needs to do is verify the correctness of the input labels and the revealed secret, which amounts to verifying hash preimages or signatures. This is something the protocol can do.

The remaining problem is the cost of the off-chain GC. For efficiency reasons, we want to verify small SNARKs on-chain like Groth16, which requires garbling expensive pairing operations and elliptic curve field arithmetic. While techniques for arithmetic garbling exist [2,4,31,34], they are not practical for this application. We need to generate a large number of garbled circuits efficiently and store them. In practice, people use Yao-based garbled circuits based on free XOR [39,50] which have garbling tables measuring 10s of gigabytes. This limits the number of garbled circuits that can be efficiently generated and the number of provers that can be practically supported. More provers makes protocols like bridging more secure, since they require at least one honest prover to function.

Another line of work [17,26], including our earlier Glock, changes the structure of the protocol to make the verifier a permissioned role. This allows the verifier to possess some secret information, in the case of Glock, and allows GCs that reveal a secret on a valid proof, in the case of BABE. Both allow simplifying the GC and reducing its size. However, making the verifier a permissioned role fundamentally alters the security of the protocol. Ideally, we would like to combine the permissionless verifier of BitVM2 [43] with garbled circuits to minimize on-chain costs without losing security. Therefore, we want to create an efficient garbled circuit for SNARK verifiers that reveals a secret on an invalid proof.

This work. We present Argo, a randomized encoding of the Groth16 verifier that releases a key when, and only when, the encoded proof is invalid. Decoding requires no secret, so anyone can evaluate it and the verifier is permissionless. The offline material is n elements of the second source group and n elements of the target group, under 21 kB for BN254 at 128-bit security. The online material consists of encodings of proof elements *over groups* using the Argo MAC encoding scheme. That is, we encode a group element A as $\phi A - K$ and call ϕ the multiplier and K the mask. This allows us to work algebraically over the groups, improving the efficiency of the scheme. We also show how to combine this with a projective Elliptic Curve MAC [18,37] which transforms an encoding of the bits of the proof to a group encoding of the proof elements. We show in Section 2 how to construct a garbling scheme for Groth16 invalidity and a BitVM protocol for permissionless, optimistic SNARK verification using Argo.

Argo adapts the techniques of other partial garbling schemes [35,34] to the group-valued setting. This is a natural and under-explored avenue for efficient randomized encodings of SNARK verifiers, and the specific problem is to encode *non-linear pairing equations*. Three ideas do the work, and Section 2 explains each of them on a single proof element. The first is to encode group elements with affine MACs whose multipliers come from a small set, replicated enough times that the vector of multipliers is a key. The second is a pairing trick: the mask of the encoding on one side of the pairing is chosen as the multiplier of

the other side times a generator, so that pairing the two encodings cancels the cross terms and yields an encoding of $e(A, B)$. Small multipliers break this, and repairing it, by contracting many small-multiplier encodings into one with a uniform multiplier and by sharing one secret across the masks on the other side, is where DDH enters. The third is that the encoding of the verification value equals its mask on a valid proof and a generator raised to the multiplier on an invalid one, so publishing the masks lets the decoder extract the key by a discrete logarithm over the small set exactly when the proof is invalid.

Randomized encodings. We describe Argo as a randomized encoding [32,2,1] with a public input, the proof, and a private input, the key, and privacy of the encoding says exactly what the application needs: the key is pseudorandom to anyone holding the encoding of a valid proof. We find this syntax convenient, but Argo could just as well be described as a garbling scheme [8,7].

Security. We prove selective security under DDH in the second source group, in the standard model. The key is the vector of multipliers, and the proof shows it is indistinguishable from a uniform vector given the transcript of a valid proof, with no loss beyond the DDH advantage. In the selective notion the proof is fixed before the offline material is published. This is natural in the case where the prover and encoder are the same party. In some applications like Bitcoin bridges the statement being proven, i.e. public input, can be influenced by other parties after setup, so we also describe an adaptive, in which the proof may depend on the offline material. We obtain it from the selective result by a standard black-box transformation in the random oracle model [7], at the cost of one additional λ -bit string in the online phase. We also show that the combined scheme requires only privacy from the projective garbling scheme. This is because when garbling affine functions, whether $ax + b \in \mathbb{F}$ or $\phi A + K \in \mathbb{G}$ with small ϕ , a break of authenticity also breaks privacy.

Related work. Our earlier work Argo MAC [18] introduced the translation from bit encodings of a curve point to a homomorphic MAC of the point, and Duty-Free Bits [37] reduced the size of that translation. Either can serve as the projective MAC in our online phase. BABE [26] adapts the techniques of Argo MAC and combines with witness encryption for linear pairing equations [25,24] to obtain small, permissioned garbled verifiers. Witness encryption for SNARK verifiers [45] aims at a related goal, releasing a message to anyone holding a valid proof, without an online phase at all. The known candidates, which build on affine determinant programs [6], appear to be broken [36,27]. Validity Argo in Section 6.3 can be read as a two-phase version of that primitive.

Other applications. We believe the techniques of Argo can be adapted to other situations where we might want to garble elliptic curve arithmetic and pairings efficiently. For example, the one-time programs of Eldridge et al. [19] are built from garbled circuits. They exploit physical hardware to simulate a kind of oblivious transfer. In order to keep the size of the input small, and reduce the amount

of hardware needed, they run Laconic OT [13] inside a garbled circuit. Recent protocols for Laconic OT [21] require pairing operations in circuit. The authors of [19] identify this cost as a significant practical impediment to their protocol. The techniques of Argo naturally adapt to other pairing-based protocols and may help reduce the cost of garbling.

2 Technical Overview

Argo fits into a large body of work on randomized encodings that exploit algebraic structure for different purposes. These work over fields and rings [32,33,14,2], lattices [2,3], and in our case elliptic curve groups. Designing schemes to work “natively” over the objects they manipulate avoids encoding those objects in less natural representations. A single group addition is simple in a group, more complex over a field, and much more complex when written as a binary circuit. This section explains the three ideas behind Argo on a single replica and a single proof element each, and closes with the application. The construction in Section 6 is nothing more than these ideas applied to every element of a Groth16 proof.

2.1 Group MACs and Replication

The basic object is an affine encoding of a group element. For A in a group of prime order p , a multiplier ϕ and a mask K , the encoding is

$$[A] = \phi A - K.$$

This is a one-time, information-theoretic MAC. The mask is uniform, so one encoding reveals nothing about ϕ , and anyone who later produces an encoding of a different element A' under the same key has computed $\phi(A' - A)$, which pins down ϕ . The mask carries no information of its own, since it can be recovered from A , ϕ and $[A]$. So ϕ is the secret and K is padding. This is the Group MAC of our earlier work Argo MAC [18], and Section 4 records the two properties we need.

In our setting we need to extract ϕ from a MAC. Publishing the mask, a technique implicit in the randomized encoding literature [12,15,16,34], leaves the decoder with ϕA and gets us some of the way there. However, we are still left with extracting a scalar from a group element, i.e. computing a discrete logarithm, which is infeasible for a general ϕ . So we take ϕ from a small set S of endomorphisms, where the discrete logarithm over S is a search over $|S|$ candidates. That makes a single encoding weak, since a guess succeeds with probability $1/|S|$. We fix this by replication. That is, encode the same element n times under independent multipliers $\phi_1, \dots, \phi_n \in S$, and the key is the whole vector ϕ . Guessing the multiplier succeeds with probability $|S|^{-n}$, so we choose n so that this is $2^{-\lambda}$. For curves with j -invariant 0 the set S consists of 0 and the six units of the endomorphism ring, so $|S| = 7$ and $n = 46$ suffices for $\lambda = 128$. Conveniently, a replicated MAC with small-set multipliers is precisely

the structure of an Argo MAC [18], which is what lets us produce the encodings from the bits of the proof in the projective instantiation.

The point of this structure is that it is linear, so it survives the operations of the verifier. Adding two encodings under the same multiplier gives an encoding of the sum, and computing the pairing of an encoded value with a public point yields an encoding of the pairing. However, to verify Groth16 we need to compute the pairing of two encoded elements which requires a new technique.

2.2 Garbling Non-linear Pairings

The Groth16 verifier is a product of pairings, and the term $e(A, B)$ pairs a proof element in $\mathbb{G}_1$ with a proof element in $\mathbb{G}_2$. We start from the standard problem of deriving an encoding of xy from encodings of x and y when the decoder knows x and y . Over a field, let $[x] = ax - b$ and $[y] = by - c$, and have the decoder compute $y[x] + [y] = a(xy) - c = [xy]$: the mask of the first encoding is the multiplier of the second, so the cross terms cancel. Now suppose we hold $[B] = \phi B - K$ in $\mathbb{G}_2$ and $[A] = \psi A - L$ in $\mathbb{G}_1$ and want an encoding of $e(A, B)$. If $y[x]$ maps to $e(A, [B])$ then $[A]$ does not live in the right group to play the role of $[y]$. Inserting a generator to compute $e([A], H)$ does not quite fix it, since $e(A, K) \neq e(\psi A, H)$ unless $K = \psi H$. It turns out this is the fix. Let $K = \psi H$, so that

$$\begin{aligned} e(A, [B]) e([A], H) &= (e(A, B)^\phi e(A, H)^{-\psi}) (e(A, H)^\psi e(L, H)^{-1}) \\ &= e(A, B)^\phi e(L, H)^{-1} = [e(A, B)], \end{aligned}$$

an encoding of $e(A, B)$ under multiplier ϕ and a mask the encoder knows in advance.

There is one remaining conceptual gap we must close for the two techniques to combine. If the encoding of A consists of $[A]_j = \psi_j A - L_j$ with small ψ_j , as it must when it comes from a projective MAC, then using $K = \psi H$ to mask B is not secure, since a mask from a small set does not hide anything. Instead, we contract the encoding of A with a public basis $\beta_1, \dots, \beta_d$ to obtain an encoding with a large multiplier:

$$[A] = \sum_j \beta_j [A]_j = \left(\sum_j \beta_j \psi_j \right) A - \sum_j \beta_j L_j = tA - L,$$

and the encoder chooses t uniformly and lets the ψ_j be its digits in the basis. Section 3.4 gives a basis for which the digit set is exactly the seven-element S and d is about $\log_7 p$.

This does not quite solve it, since B is encoded n times, once per replica, as $[B]_i = \phi_i B - K_i$, and we cannot use the same mask in each. Instead we use a different U_i for each and let $K_i = tU_i$. The same t appears in every mask, because the cancelling term $e([A], U_i)$ has multiplier t for every i . So the masks are $tU_1, \dots, tU_n$ for public $U_i = r_i H$, which are n ElGamal encryptions under the public key tH . Hiding the n multipliers of B behind them is exactly the

DDH assumption in $\mathbb{G}_2$, and it is the only place any assumption enters. The other encodings are padded with fresh uniform masks and hide their multipliers unconditionally. The decoder never sees t : it pairs $[A]$ with the public U_i , and the term $e(A, U_i)^t$ that would expose t is cancelled by the same term inside $e(A, [B]_i)$.

2.3 Releasing a Key on an Invalid Proof

Applying the trick to every term of the verifier, the decoder obtains, for each replica, an encoding of the verification value $E(\pi)$ under multiplier ϕ_i and a mask p_i that the encoder computed in the offline phase:

$$[E(\pi)]_i = E(\pi)^{\phi_i} p_i^{-1}.$$

On a valid proof, $E(\pi) = 1$ and the encoding is just the mask. This is, in essence, the algebraic trick that drives witness encryption for linear pairing equations. On an invalid proof the encoding is $E(\pi)^{\phi_i} p_i^{-1}$, which in general leaks neither ϕ_i nor p_i . The encoder therefore publishes the masks offline. On a valid proof the decoder then learns exactly p_i , which it already knew. On an invalid proof, $E(\pi)$ is a generator of the target group, the decoder holds $E(\pi)^{\phi_i}$, and a search over S recovers ϕ_i . Doing this for all n replicas yields the key. The same transcript therefore hides the key or reveals it depending only on the validity of the proof.

2.4 Our Contributions

The main novel contribution of Argo is leaking a secret when a non-linear pairing equation is not satisfied, i.e. on an invalid proof. Our work fits into the larger body of work on partial garbling schemes and randomized encodings [32, 2, 35, 34] that exploit algebraic structure. We show these techniques translate to the group-valued setting efficiently. We use the encoding of Argo MAC both to leak a secret from an invalid proof and for compatibility with projective garbling schemes. Other work [26] builds on Argo MAC to release a secret from a valid proof, but these techniques do not obviously translate to the non-linear pairing setting. Our proof for the valid case is also novel: it shows security for leaking a secret from a valid proof under DDH in the standard model, where the proof of [26] for that setting is in the generic bilinear group and random oracle models.

Argo contributes three things. The pairing composition of Section 4, which cancels a secret shared between a DDH mask in one source group and a decomposed multiplier in the other, so that the decoder obtains an encoding of the verification value with $3n$ pairings and no garbled pairing at all. The use of small-set multipliers as the key itself, so that an invalid verification value leaks the key by a discrete logarithm the decoder can afford, while a valid one leaks nothing. And the proof that these hide the key under DDH alone, in the standard model, together with the observation in Section 8 that for affine MACs privacy of the garbling already implies its authenticity.

2.5 Argo BitVM

Argo is built for optimistic SNARK verification on Bitcoin. In a BitVM game, there are three roles. The *prover* claims to hold a valid Groth16 proof for a public statement. A *verifier* is anyone who wants to submit a fraud proof for an invalid claim. The *chain* is Bitcoin, which can check a hash preimage and a signature and nothing else. At *setup*, the prover runs the offline phase, publishes the elements U_i , the masks p_i and the garbling table of the projective MAC off chain, commits on chain to $\text{Hash}(k)$ for a preimage-resistant hash Hash , and locks collateral. At *claim* time it publishes the labels for the bits of its proof, as in the assertion transaction of BitVM3 [48], together with the key κ that decrypts the offline material in the adaptive variant. In a *dispute*, any verifier evaluates the table on the labels, runs the decoder, and, if the proof is invalid, publishes k . The chain checks the preimage and the collateral is forfeited. The disproof is a few dozen bytes and can be published by anyone, which is why the verifier is permissionless. Glock [17] and BABE [26] give this up: in those schemes the evaluator holds decoding information that cannot be public and must be fixed at setup.

To instantiate Argo in BitVM, we need to compose it with a projective garbling scheme for the group encodings like Argo MAC or Duty-Free Bits. We show that this requires only privacy from the garbling scheme in Section 8. We prove that the transcript of a valid proof is indistinguishable from one generated with an independent uniform key, selectively under DDH in the standard model in Section 7 and adaptively in the random oracle model in Section 7.2, and that the transcript of an invalid proof reveals the key and nothing else. To fraudulently claim the prover’s collateral, a malicious verifier would need to invert Hash on a uniform key with $n \log_2 |S| \geq \lambda$ bits of entropy.

To make this protocol practical, we also need to address malicious security and validity of the curve points input to the projective garbling scheme. The former can be addressed in a black-box way using cut-and-choose [41,40], and which Mosaic [38], coauthored by one of us, makes efficient in Bitcoin. They use signatures and verifiable secret sharing to ensure that only one set of labels need be published on-chain. Curve point validity can be addressed using techniques present in Argo MAC, but essentially amounts to garbling the curve equation. This is cheap relative to the rest of the protocol, but in the case of an invalid point we need to ensure that it leaks the secret. We leave a proof of the end-to-end protocol to future work.

Table 1 compares Argo with the schemes closest to it. The Yao row is the BitVM3 protocol with a Boolean garbling of the Groth16 verifier using free XOR and privacy-free half gates [39,50], and the Argo row is the same protocol with Argo and the projective MAC in place of the Boolean circuit. The off-chain size for Argo is the offline material of the encoding plus the tables of the projective MAC, and the on-chain costs are those of the BitVM3 transaction flow, which Argo does not change.

The BitVM2 figures are from [43], which reports claim and disproof transactions above Bitcoin’s 400 kB standardness limit and disproof scripts between 0.5

Table 1. Optimistic Groth16 verification on Bitcoin. Off-chain size is the material the prover publishes at setup, and garbling is the single-threaded time to produce it, both for a single instance with honest setup. On-chain cost is the size of the claim and of a disproof. A dash means the figure is not reported.

Scheme	Verifier	Off-chain	Garbling	Claim	Disproof
BitVM2 [43]	anyone	none	none	> 400 kB	0.5–4 MB
Yao [48,10]	anyone	≈ 43 GB	≈ 6 min	≈ 56 kB	≈ 200 B
Glock [17]	fixed	> 100 MB	–	as Yao	as Yao
BABE [26]	fixed	≈ 22 MB	≈ 175 ms	as Yao	as Yao
Argo	anyone	21 kB + 25 MB	≈ 140 ms	as Yao	as Yao

and 4 MB; the total on-chain footprint of a dispute in its mainnet experiment was over 5 MB [26]. The Yao off-chain size is for the Boolean Groth16 verifier of [10], which has 10.4 billion gates of which 2.7 billion are not free, at one 16-byte ciphertext per non-free gate under privacy-free half gates; this agrees with the 42 GiB reported in [26]. Its on-chain figures are from [48]. The BABE size and its setup time are from [26], which also reports the six minutes for garbling the Boolean verifier on a single core; the repository itself reports a little over three minutes with full parallelism. The Argo garbling time is the single-threaded total of Table 2. The Glock size is our estimate for the construction of [17], which does not state a figure.

3 Preliminaries

3.1 Notation

We write λ for the security parameter and $[n]$ for the set $\{1, \dots, n\}$. For a finite set X , we write $x \leftarrow X$ for a uniformly random sample. We denote groups as $\mathbb{G}$, which always have prime order p and fields as $\mathbb{F} = \mathbb{F}_p$. We write Hash for a preimage resistant hash function. The hash function is preimage resistant if given $\text{Hash}(k)$ for $k \leftarrow D$, no PPT outputs k' with $\text{Hash}(k') = \text{Hash}(k)$ except with negligible probability. The encoding itself does not use a hash function, but it is used in the composite protocol. The application commits to the key with Hash , and the adaptive transformation of Section 7.2 uses a separate function $\mathcal{O}$ modeled as a programmable random oracle.

3.2 Bilinear Groups

A bilinear group is a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of prime order p , and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a nondegenerate bilinear map. Following convention, we write $\mathbb{G}_1$ and $\mathbb{G}_2$ additively and $\mathbb{G}_T$ multiplicatively. The element G is the canonical generator of $\mathbb{G}_1$, H is the canonical generator of $\mathbb{G}_2$, and $g = e(G, H)$ is the canonical generator of $\mathbb{G}_T$. Bilinearity means that $e(aP, bQ) = e(P, Q)^{ab}$ for all $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and $a, b \in \mathbb{F}$. We assume a Type-3 pairing [22], which is also assumed by Groth16, so $\mathbb{G}_1 \neq \mathbb{G}_2$.

and further there is no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Since each group has prime order, and because the curves are not supersingular, the endomorphism ring is isomorphic to $\mathbb{F}$ and every endomorphism is multiplication by a scalar. For certain elliptic curves, i.e. with complex multiplication, some endomorphisms are much cheaper to compute than a generic scalar multiplication. When the curve has j -invariant 0, multiplication by a primitive third root of unity ω can be computed as a linear function of the Weierstrass coordinates with a single field multiplication [23]. Since the curve is not supersingular, $\omega \in \mathbb{F}$ and multiplication by ω is an endomorphism of each of the three groups. This case occurs in practice, since pairing-friendly elliptic curves like BN254 typically have complex multiplication.

Definition 1 (DDH). *The decisional Diffie–Hellman assumption [11] holds in a group $\mathbb{G}$ with generator P if the distributions*

$$(aP, bP, abP) \quad \text{and} \quad (aP, bP, cP)$$

are computationally indistinguishable, where $a, b, c \leftarrow \mathbb{F}$.

We only need DDH in the second source group $\mathbb{G}_2$, which follows from standard assumptions [29] for curves such like BN254 and BLS12-381. It holds in the generic bilinear group model in which Groth16 was originally proven knowledge sound [28]. While DDH and Groth16 knowledge soundness are not directly comparable as security assumptions, we believe that the security assumptions of Argo are comparable to those underlying the security of Groth16 and other pairing-based SNARKs.

3.3 Groth16

Groth16 [28] is a small pairing-based SNARK. We will not depend on any of the specific details of the protocol except that its verifier is a single pairing product equation. A proof system consists of algorithms **Setup**, **Prove** and **Verify**. The setup takes a relation and outputs a proving key and a verifier key vk . The prover takes the proving key, a public input and a witness and outputs a proof. The verifier takes vk , the public input and the proof and accepts or rejects.

We consider a single public input $x \in \mathbb{F}$, and we treat it as part of the proof. The verifier key is

$$\text{vk} = (G_1, H_1, H_2, h_3) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_2 \times \mathbb{G}_T,$$

and a proof is $\pi = (A, B, C, x) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \times \mathbb{F}$. We define the verification function

$$E(\pi) = e(A, B) e(C, H_1) e(xG_1, H_2) h_3 \in \mathbb{G}_T,$$

and the verifier accepts π if and only if $E(\pi) = 1$. We say π is valid when $E(\pi) = 1$ and invalid otherwise.

In Groth's notation the verifier checks $e(A, B) = e(\alpha, \beta) e(L_0 + xL_1, \gamma) e(C, \delta)$. Moving every term to one side gives our form with $H_1 = -\delta$, $H_2 = -\gamma$, $G_1 = L_1$ and $h_3 = e(\alpha, \beta)^{-1} e(L_0, \gamma)^{-1}$. Groth16 also supports multiple inputs generalizing the term $e(xG_1, H_2)$ to $\prod_\ell e(x_\ell G_{1,\ell}, H_2)$. Our protocol efficiently extends to multiple inputs, but we focus on the single input version for clarity.

3.4 Decompositions

Let $S \subseteq \mathbb{F}$ be a finite set of endomorphisms containing 0, and let $\beta_1, \dots, \beta_d \in \mathbb{F}$ be public. We assume an algorithm `Decompose` : $\mathbb{F} \rightarrow S^d$ that on input $t \in \mathbb{F}$ returns $\psi = (\psi_1, \dots, \psi_d)$ with

$$t = \sum_{j=1}^d \beta_j \psi_j.$$

Counting shows that $|S|^d \geq p$, so $d \geq \log_{|S|} p$. Two examples matter to us.

- **Binary.** $S = \{0, 1\}$ and $\beta_j = 2^{j-1}$. Then `Decompose` returns the bits of t and $d = \lceil \log_2 p \rceil$.
- **Curves with j -invariant 0.** Let $\omega \in \mathbb{F}$ be a primitive cube root of unity. We can write an element $x \in \mathbb{F}$ using digits $S = \{0, \pm 1, \pm \omega, \pm \omega^2\}$ and base $\beta_i = (3 + \omega)^i$. To compute the decomposition, first we compute integers a, b with $t \equiv a + b\omega \pmod{p}$ and $|a|, |b| = O(\sqrt{p})$ by the GLV lattice reduction [23], and set $z = a + b\omega \in \mathbb{Z}[\omega]$. Then we exploit the fact that $a + b\omega = a - 3b \pmod{3 + \omega}$ and that $3 + \omega \mid 7$. Each element of S maps to a different element mod 7 under this map, so we can read off the low order digit $\psi \in S$. Then, subtract off ψ and divide by $3 + \omega$. Repeat until the value vanishes. Since the division is exact and reduces the norm by 7, the algorithm terminates in about $\log_7 p$ with a maximum number of 92 iterations for BN254.

3.5 Randomized Encodings

A randomized encoding [32, 2, 1] of a function f is a randomized function $\hat{f}$ that is simpler than f but reveals the same information. We use the partial form of Ishai and Wee [35], where the function takes a public input x and a private input z . The encoding $\hat{f}(x, z; \rho)$ over randomness ρ comes with a decoder `Decode` and a simulator `Sim` such that

- **Correctness.** $\text{Decode}(\hat{f}(x, z; \rho), x) = f(x, z)$ for every x, z and ρ .
- **Privacy.** The distributions $\hat{f}(x, z; \rho)$ over ρ and $\text{Sim}(x, f(x, z))$ are indistinguishable for every x and z .

The decoder and the simulator both have x , so the encoding is not required to hide it. Privacy says the encoding reveals nothing about z beyond $f(x, z)$. It can be perfect, statistical or computational, and we work with computational privacy throughout.

We use encodings with an offline part and an online part. The encoding is $\hat{f}(x, z; \rho) = (\hat{f}_{\text{off}}(z; \rho), \hat{f}_{\text{on}}(x, z; \rho))$, where the offline part depends on the private input and the randomness and can be published before x is known. For example in garbled circuits, the garbled circuit is the offline part, the input labels are the online part, and the private input is the garbler's secrets [8]. The privacy notion then splits as [7].

- **Selective privacy.** The public input x is fixed before the offline part is published. The simulator receives x and $f(x, z)$ together and produces both parts at once. This is the notion above.
- **Adaptive privacy.** The adversary sees the offline part and then chooses x . The simulator must produce the offline part before it learns anything, and then produce the online part given x and $f(x, z)$.

Adaptive privacy is strictly stronger, and it is well known to be hard to achieve for garbled circuits in the standard model [7]. However, it follows from selective privacy in the random oracle model by encrypting the offline part under a key revealed online. Section 5 specializes both notions to the verifier, and Section 7.2 applies the transformation.

4 Group MACs

A Group MAC is an affine function of a group element generalizing information-theoretic MACs [47,9]. We recall the definition from our earlier work Argo MAC [18] and record the two properties we need: MACs are homomorphic, and the pairing combines a MAC in $\mathbb{G}_1$ with a MAC in $\mathbb{G}_2$ into a MAC in $\mathbb{G}_T$.

Definition 2 (Group MAC). *A Group MAC is parameterized by $(\mathbb{G}, S, n)$, where $\mathbb{G}$ is a group of prime order p , $S \subseteq \mathbb{F}$ is a set of endomorphisms of $\mathbb{G}$, and $n \geq 1$ is a replication factor. A key is a pair $\text{key} = (\phi, K) \in S^n \times \mathbb{G}^n$. The MAC of $A \in \mathbb{G}$ under key is the vector $[A]_{\text{key}} = ([A]_1, \dots, [A]_n)$ with*

$$[A]_i = \phi_i A - K_i \quad \text{for } i \in [n].$$

We write $[A]$ when the key is clear from context. When $\mathbb{G}$ is written multiplicatively, as for $\mathbb{G}_T$, the same definition reads $[m]_i = m^{\phi_i} K_i^{-1}$.

We call ϕ the multiplier and K the mask. The multiplier is the only part of the key that carries information about A , since the mask can be recovered from A , ϕ and $[A]$. Recovering ϕ from A , K and $[A]$ is a discrete logarithm over the set S , which is easy only when S is small. The parameter n controls how hard it is to guess ϕ . We choose n so that $|S|^n \geq 2^\lambda$.

Lemma 1 (Homomorphism). *Let $[A]$ and $[A']$ be MACs under keys (ϕ, K) and (ϕ, K') with the same multiplier. Then $[A] + [A']$ is the MAC of $A + A'$*

under $(\phi, K + K')$. Let $[A]_1, \dots, [A]_d$ be MACs of the same element A under keys (ψ_j, K_j) , where the multipliers may differ, and let $\beta_1, \dots, \beta_d \in \mathbb{F}$. Then

$$\sum_{j=1}^d \beta_j [A]_j$$

is the MAC of A under the key $(\sum_j \beta_j \psi_j, \sum_j \beta_j K_j)$.

Proof. Both claims follow directly from the definition, since $\sum_j \beta_j (\psi_j A - K^{(j)}) = (\sum_j \beta_j \psi_j) A - \sum_j \beta_j K^{(j)}$. $\square$

Note that in the second case the resulting multiplier $\sum_j \beta_j \psi_j$ is not necessarily an element of S . For appropriate d, S it can be any field element, which we exploit in the protocol.

Lemma 2 (Unforgeability). *Let $\text{key} = (\phi, K) \leftarrow S^n \times \mathbb{G}^n$. Fix $A \in \mathbb{G}$ and $A' \neq A$. Any algorithm given A and $[A]_{\text{key}}$ outputs $[A']_{\text{key}}$ with probability at most $|S|^{-n}$.*

Proof. Since K is uniform, $[A]$ is uniform in $\mathbb{G}^n$ and independent of ϕ . A forgery satisfies $[A']_i - [A]_i = \phi_i(A' - A)$ for every i . Since $A' - A \neq 0$ has order p and distinct elements of S are distinct scalars, the map $\phi_i \mapsto \phi_i(A' - A)$ is injective. Producing the forgery therefore determines ϕ , which is uniform in S^n and independent of the algorithm's view. $\square$

Restricted Group MACs. The mask K in a Group MAC is uniform in $\mathbb{G}^n$. A Restricted Group MAC instead samples K from a distribution D over $\mathbb{G}^n$. The MAC is then no longer statistically hiding, and its security depends on D . The case we use is the following.

Definition 3 (DDH MAC). *A DDH MAC is a Restricted Group MAC with public $U = (U_1, \dots, U_n) \in \mathbb{G}^n$ and mask $K_i = tU_i$ for a secret $t \leftarrow \mathbb{F}$ shared across all $i \in [n]$. That is,*

$$[A]_i = \phi_i A - tU_i.$$

We use it in $\mathbb{G}_2$. When $U_i = r_i H$, the pair $(U_i, [B]_i) = (r_i H, \phi_i B - r_i P)$ with $P = tH$ is exactly an ElGamal encryption [20] of $\phi_i B$ under the public key P with randomness r_i . This is the source of the name, and of the DDH assumption in our security proof. The encoder knows both t and the r_i .

The reason to share t across the replicas of a DDH MAC is the following lemma, which is the heart of the construction. A DDH MAC in $\mathbb{G}_2$ and an ordinary MAC in $\mathbb{G}_1$ with multiplier t combine under the pairing into a MAC in $\mathbb{G}_T$ whose multiplier is ϕ_i again.

Lemma 3 (Pairing composition). *Let $U_i = r_i H \in \mathbb{G}_2$ and let $[B]_i = \phi_i B - tU_i$ be a DDH MAC of $B \in \mathbb{G}_2$. Let $[A] = tA - sG \in \mathbb{G}_1$ be a MAC of $A \in \mathbb{G}_1$ under the key (t, sG) . Then*

$$e(A, [B]_i) e([A], U_i) = e(A, B)^{\phi_i} g^{-sr_i},$$

which is a MAC of $e(A, B) \in \mathbb{G}_T$ under the key (ϕ_i, g^{sr_i}) .

Proof. By bilinearity,

$$e(A, [B]_i) = e(A, B)^{\phi_i} e(A, U_i)^{-t} \quad \text{and} \quad e([A], U_i) = e(A, U_i)^t e(G, U_i)^{-s}.$$

The cross terms $e(A, U_i)^{\pm t}$ cancel and $e(G, U_i)^{-s} = g^{-sr_i}$. $\square$

The decoder never learns t . It only ever pairs the MAC $[A]$ with the public U_i , and the term $e(A, U_i)^t$ that would otherwise expose t is cancelled by the corresponding term in $e(A, [B]_i)$.

5 Randomized Encodings of a Verifier

We describe the scheme as a randomized encoding, in the sense of Section 3.5, with public input a proof π , private input a key k from a key space $\mathcal{K}$, and

$$f(\pi, k) = \begin{cases} k & \text{if } E(\pi) \neq 1, \\ \perp & \text{if } E(\pi) = 1. \end{cases}$$

The proof is public because the decoder needs it, and the key is the only secret. The offline message is published before the proof is known, the online message encodes a specific proof, and the pair decodes to k if and only if the proof is invalid. For Argo the key space is $\mathcal{K} = S^n$. A secret of any other form can be released by encrypting it under a key derived from k and publishing the ciphertext in the offline phase.

Definition 4 (Verifier encoding). *An encoding scheme for a verifier with key $\mathbf{vk}$ and key space $\mathcal{K}$ consists of three algorithms.*

- $\text{Offline}(\mathbf{vk}, k) \rightarrow (\sigma, \text{st})$. Takes a key $k \in \mathcal{K}$ and outputs a public offline encoding σ and a private state st .
- $\text{Online}(\text{st}, \pi) \rightarrow \tau$. Outputs the online encoding τ of a proof π .
- $\text{Decode}(\mathbf{vk}, \sigma, \tau, \pi) \rightarrow k'$. Outputs a key $k' \in \mathcal{K}$ or $\perp$.

The scheme is correct if for every $\mathbf{vk}$ and every key $k \in \mathcal{K}$, the decoder returns k on every invalid proof and $\perp$ on every valid proof, with probability 1 over the randomness of Offline and Online.

Security is the privacy notion of Section 3.5 for the function f , with the adversary choosing both inputs. It says that the encodings of a valid proof reveal nothing about k , and that the encodings of an invalid proof reveal nothing beyond k . In the selective notion the proof is fixed before the offline encoding is published. In the adaptive notion the adversary chooses the proof after seeing the offline encoding. In both, the key is fixed first, since the offline encoding depends on it.

Definition 5 (Selective security). *An encoding scheme is selectively secure if there is a simulator Sim such that for every efficient adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ and every $\mathbf{vk}$, the outputs of the following two experiments are computationally indistinguishable.*

Real_A(vk): $(k, \pi, \text{st}_A) \leftarrow \mathcal{A}_0(\text{vk})$ $(\sigma, \text{st}) \leftarrow \text{Offline}(\text{vk}, k)$ $\tau \leftarrow \text{Online}(\text{st}, \pi)$ $\text{return } \mathcal{A}_1(\text{st}_A, \sigma, \tau)$	Ideal_{A, Sim}(vk): $(k, \pi, \text{st}_A) \leftarrow \mathcal{A}_0(\text{vk})$ $(\sigma, \tau) \leftarrow \text{Sim}(\text{vk}, \pi, f(\pi, k))$ $\text{return } \mathcal{A}_1(\text{st}_A, \sigma, \tau)$
--	---

Definition 6 (Adaptive security). *An encoding scheme is adaptively secure if there is a simulator $\text{Sim} = (\text{Sim}_0, \text{Sim}_1)$ such that for every efficient adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ and every vk , the outputs of the following two experiments are computationally indistinguishable.*

Real_A(vk): $(k, \text{st}_A) \leftarrow \mathcal{A}_0(\text{vk})$ $(\sigma, \text{st}) \leftarrow \text{Offline}(\text{vk}, k)$ $(\pi, \text{st}_A) \leftarrow \mathcal{A}_1(\text{st}_A, \sigma)$ $\tau \leftarrow \text{Online}(\text{st}, \pi)$ $\text{return } \mathcal{A}_2(\text{st}_A, \tau)$	Ideal_{A, Sim}(vk): $(k, \text{st}_A) \leftarrow \mathcal{A}_0(\text{vk})$ $(\sigma, \text{st}_{\text{Sim}}) \leftarrow \text{Sim}_0(\text{vk})$ $(\pi, \text{st}_A) \leftarrow \mathcal{A}_1(\text{st}_A, \sigma)$ $\tau \leftarrow \text{Sim}_1(\text{st}_{\text{Sim}}, \pi, f(\pi, k))$ $\text{return } \mathcal{A}_2(\text{st}_A, \tau)$
---	--

We prove selective security in the standard model, under DDH, in Section 7. This is the case where the proof is fixed independently of the offline encoding and can not be adaptively chosen by the adversary. We obtain adaptive security by a black-box transformation in the random oracle model in Section 7.2. The protocol of Section 2.5 uses the adaptive variant.

6 The Argo Encoding

Fix a bilinear group, a verifier key $\text{vk} = (G_1, H_1, H_2, h_3)$, a set S with a decomposition $(\beta_j)_{j \in [d]}$, and a replication factor n . The encoder MACs each element of the proof, and the decoder uses the pairing to turn these MACs into a MAC of $E(\pi)$ under a key (ϕ_i, p_i) that the encoder knows in advance. The encoder publishes the masks p_i in the offline phase, so the decoder can strip them and obtain $E(\pi)^{\phi_i}$. When the proof is invalid, $E(\pi) \neq 1$, and the decoder recovers ϕ_i by searching the small set S . When the proof is valid, $E(\pi)^{\phi_i} = 1$, and the decoder learns nothing about ϕ_i . The key is the vector $\phi = (\phi_1, \dots, \phi_n)$ itself, so the key space is S^n .

The multipliers ϕ_i of the MACs on B , C and x lie in the small set S . The MAC of A must have the uniform multiplier t cancel against the DDH MAC on B . We achieve this using d MACs with multipliers in S using **Decompose** and Lemma 1. We put the DDH MAC on B and the decomposition on A rather than the other way around because the decomposition needs $d > n$ replicas, and in the projective instantiation a replica in $\mathbb{G}_2$ costs about twice a replica in $\mathbb{G}_1$.

6.1 Construction

Offline phase. $\text{Offline}(\text{vk}, k)$ with $k = (\phi_1, \dots, \phi_n) \in S^n$ does the following.

1. Sample $t \leftarrow \mathbb{F}$ and $s_j \leftarrow \mathbb{F}$ for $j \in [d]$, and set $s = \sum_j \beta_j s_j$.

2. Sample $r_i, v_i, w_i \leftarrow \mathbb{F}$ for $i \in [n]$.
3. Compute $\psi = \text{Decompose}(t)$, so that $t = \sum_j \beta_j \psi_j$ with $\psi_j \in S$.
4. Compute $U_i = r_i H \in \mathbb{G}_2$ for $i \in [n]$.
5. Compute

$$p_i = h_3^{\phi_i} g^{sr_i} e(G, H_1)^{v_i} e(G_1, H_2)^{w_i} \in \mathbb{G}_T \quad \text{for } i \in [n].$$

6. Output $\sigma = (U_1, \dots, U_n, p_1, \dots, p_n)$ and the state $\mathbf{st}$ consisting of t, ψ , the s_j , and the ϕ_i, r_i, v_i, w_i .

Online phase. $\text{Online}(\mathbf{st}, \pi)$ with $\pi = (A, B, C, x)$ outputs $\tau = ([A], [B], [C], [x])$, where

$$\begin{aligned} [A]_j &= \psi_j A - s_j G \in \mathbb{G}_1 & \text{for } j \in [d], \\ [B]_i &= \phi_i B - t U_i \in \mathbb{G}_2 & \text{for } i \in [n], \\ [C]_i &= \phi_i C - v_i G \in \mathbb{G}_1 & \text{for } i \in [n], \\ [x]_i &= \phi_i x - w_i \in \mathbb{F} & \text{for } i \in [n]. \end{aligned}$$

Here $[B]$ is a DDH MAC with mask $t U_i$, $[C]$ and $[x]$ are Group MACs with uniform masks, and $[A]_j$ is a Group MAC with multiplier ψ_j .

Decoding. $\text{Decode}(\mathbf{vk}, \sigma, \tau, \pi)$ does the following.

1. If $E(\pi) = 1$, output $\perp$.
2. Compute $[A] = \sum_{j=1}^d \beta_j [A]_j \in \mathbb{G}_1$.
3. For each $i \in [n]$ compute

$$[E(\pi)]_i = e(A, [B]_i) e([A], U_i) e([C]_i, H_1) e(G_1, H_2)^{[x]_i} \in \mathbb{G}_T$$

and $Y_i = [E(\pi)]_i \cdot p_i$.

4. For each $i \in [n]$ find $\phi'_i \in S$ with $E(\pi)^{\phi'_i} = Y_i$ by trying every element of S . If no such element exists, output $\perp$.
5. Output $k' = (\phi'_1, \dots, \phi'_n)$.

Step 4 is a discrete logarithm over the set S , and it is only efficient because S is small. The base $E(\pi)$ is the same for every i , so the decoder computes the $|S|$ values $E(\pi)^\phi$ for $\phi \in S$ once and then does n table lookups.

6.2 Correctness

Lemma 4 (Correctness). *For every proof π , the values computed by the decoder satisfy*

$$[E(\pi)]_i = E(\pi)^{\phi_i} p_i^{-1} \quad \text{and} \quad Y_i = E(\pi)^{\phi_i} \quad \text{for all } i \in [n].$$

That is, $[E(\pi)]$ is a Group MAC of $E(\pi) \in \mathbb{G}_T$ under the key (ϕ, p) . In particular, if π is invalid then Decode outputs k .

Proof. By Lemma 1, $[A] = tA - sG$ is a MAC of A under the key (t, sG) . By Lemma 3,

$$e(A, [B]_i) e([A], U_i) = e(A, B)^{\phi_i} g^{-sr_i}.$$

By bilinearity, the remaining two factors are

$$\begin{aligned} e([C]_i, H_1) &= e(C, H_1)^{\phi_i} e(G, H_1)^{-v_i}, \\ e(G_1, H_2)^{[x]_i} &= e(xG_1, H_2)^{\phi_i} e(G_1, H_2)^{-w_i}. \end{aligned}$$

Multiplying everything together and inserting $h_3^{\phi_i} h_3^{-\phi_i}$ gives

$$\begin{aligned} [E(\pi)]_i &= (e(A, B) e(C, H_1) e(xG_1, H_2) h_3)^{\phi_i} \cdot (h_3^{\phi_i} g^{sr_i} e(G, H_1)^{v_i} e(G_1, H_2)^{w_i})^{-1} \\ &= E(\pi)^{\phi_i} p_i^{-1}, \end{aligned}$$

and multiplying by p_i gives $Y_i = E(\pi)^{\phi_i}$. If π is invalid then $E(\pi) \neq 1$ generates a subgroup of order p , and distinct elements of S are distinct scalars modulo p , so $\phi \mapsto E(\pi)^\phi$ is injective on S . The search in step 4 therefore returns $\phi'_i = \phi_i$ for every i , and the decoder outputs k . $\square$

When the proof is valid, $Y_i = 1$ for every i and reveals nothing. The multipliers are then only present in $[A]$, $[C]$, $[x]$ and p , each of which is masked. The proof in Section 7 shows that under DDH the transcript of a valid proof is indistinguishable from one generated with an independent key. Guessing the key then succeeds with probability $|S|^{-n}$, which is why n is chosen so that $|S|^n \geq 2^\lambda$.

Note that $p_i = h_3^{\phi_i} \cdot m_i$ for a mask m_i , so p_i is itself a Group MAC of h_3 under the key (ϕ_i, m_i^{-1}) . Publishing the masks amounts to publishing a MAC of the constant term of the verification equation in the offline phase.

6.3 Validity Argo

The same encoding can release a key on a valid proof instead. The offline phase no longer publishes the masks, so $\sigma = (U_1, \dots, U_n)$, and the released value is the vector of masks $(p_1, \dots, p_n)$. The decoder computes $[E(\pi)]_i$ as before and outputs $([E(\pi)]_1^{-1}, \dots, [E(\pi)]_n^{-1})$. By Lemma 4 this is the mask vector when π is valid. When π is invalid, recovering p_i requires dividing out $E(\pi)^{\phi_i}$, and hence guessing ϕ_i . This variant encapsulates a key rather than encoding a chosen one, since the masks are determined by the encoder's randomness, and the released vector is unpredictable rather than uniform, since it takes only $|S|^n$ values given the transcript. A uniform key is obtained by hashing it. The variant is adaptively secure under DDH directly, because its offline message is independent of the pads. We prove this in Proposition 1.

6.4 Efficiency

The offline encoding is n elements of $\mathbb{G}_2$ and n elements of $\mathbb{G}_T$. The online encoding is $d + n$ elements of $\mathbb{G}_1$, n elements of $\mathbb{G}_2$ and n elements of $\mathbb{F}$. The

Table 2. Cost of the projective MAC for each proof element, from Argo MAC [18], Table 1, for BN254 at 128-bit security with $|S| = 7$, single-threaded on an Apple M4.

Element	Group and replicas	Garbling	Evaluation	Table size
A	$\mathbb{G}_1, d = 92$	45 ms	22 ms	7.1 MB
B	$\mathbb{G}_2, n = 46$	72 ms	37 ms	13.9 MB
C	$\mathbb{G}_1, n = 46$	23 ms	11 ms	3.6 MB
Total		140 ms	70 ms	24.6 MB

decoder computes $[A]$ with d endomorphism applications and additions in $\mathbb{G}_1$, then $3n$ pairings and n exponentiations in $\mathbb{G}_T$, since $e(G_1, H_2)$ is a constant that can be precomputed, and finally $|S|$ exponentiations in $\mathbb{G}_T$ and n table lookups for the discrete logarithms over S .

For BN254 with $\lambda = 128$ and the j -invariant 0 choice of S , we have $|S| = 7$, $n = \lceil 128 / \log_2 7 \rceil = 46$ and $d = 92$. With 32-byte field elements, compressed $\mathbb{G}_1$ and $\mathbb{G}_2$ elements of 32 and 64 bytes, and $\mathbb{G}_T$ elements of 384 bytes, the offline encoding is about 21 kB and the online encoding is about 9 kB. The decoder performs 138 pairings. The offline phase of the encoding itself costs n exponentiations in $\mathbb{G}_T$ and n scalar multiplications in $\mathbb{G}_2$, and the online phase is affine in the proof, so both are negligible next to the projective MAC.

The cost of the composed scheme is dominated by the projective MAC tables. Argo MAC [18] reports garbling time, evaluation time and table size for BN254 at $\lambda = 128$ with $|S| = 7$, single-threaded on an Apple M4, for exactly the replica counts we use. Table 2 collects the rows we need. The encoding of A uses $d = 92$ replicas in $\mathbb{G}_1$, the encoding of B uses $n = 46$ replicas in $\mathbb{G}_2$, and the encoding of C uses $n = 46$ replicas in $\mathbb{G}_1$. The encoding of x is a MAC over $\mathbb{F}$ and is negligible. In total the tables are about 25 MB, and the single-threaded times sum to about 140 ms for garbling and 70 ms for evaluation. The three encodings are independent, and each is itself embarrassingly parallel across replicas, so with a few threads the wall-clock time for either is tens of milliseconds. The 21 kB of Argo offline material is three orders of magnitude smaller than the tables, and moving the decomposition from B to A saves about 10 MB relative to the opposite assignment.

7 Security

7.1 Selective Security

We start by showing the standard reduction of DDH over n pairs using the same secret t to a single instance of DDH. This uses the random self-reducibility of DDH [11].

Lemma 5 (n -fold DDH). *If DDH holds in $\mathbb{G}_2$, then for every polynomial n the distributions*

$$(tH, (r_i H, tr_i H)_{i \in [n]}) \quad \text{and} \quad (tH, (r_i H, R_i)_{i \in [n]})$$

are computationally indistinguishable, where $t, r_i \leftarrow \mathbb{F}$ and $R_i \leftarrow \mathbb{G}_2$.

Proof. Given a triple (P, U, Z) , we can extend it to n points by sampling n pairs of random scalars (a_i, b_i) and letting $U_i = a_i U + b_i H$ and $Z_i = a_i Z + b_i P$. In the case the triple is DDH, i.e. $P = tH$ and $Z = tU$, then all the triples are DDH, i.e. $Z_i = tU_i$. When Z is uniform then except with probability $1/p$, which occurs when Z happens to form a DDH triple, (U, Z) and (H, P) are linearly independent and so (U_i, Z_i) are uniform and independent. $\square$

The proof has one algorithm and one reduction. The algorithm is an encoder that takes the key, the proof and n pairs (U_i, Z_i) , and never uses t . On real DDH pairs it produces the honest transcript. On random pairs it produces the simulated one.

Encoding from pairs. $\text{Encode}^*(\text{vk}, k, \pi, (U_i, Z_i)_{i \in [n]})$ with $k = (\phi_1, \dots, \phi_n)$ and $\pi = (A, B, C, x)$ does the following.

1. Set $[B]_i = \phi_i B - Z_i \in \mathbb{G}_2$ for $i \in [n]$.
2. Sample $[A]_j \leftarrow \mathbb{G}_1$ for $j \in [d]$, and $[C]_i \leftarrow \mathbb{G}_1$ and $[x]_i \leftarrow \mathbb{F}$ for $i \in [n]$.
3. Compute $[A] = \sum_j \beta_j [A]_j$ and, for $i \in [n]$,

$$\Pi_i = e(A, [B]_i) e([A], U_i) e([C]_i, H_1) e(G_1, H_2)^{[x]_i}$$

and $p_i = E(\pi)^{\phi_i} \Pi_i^{-1}$.

4. Output $\sigma = (U, p)$ and $\tau = ([A], [B], [C], [x])$.

The algorithm never uses any of the scalar randomness, t, ψ, s, v or w , and it does not run **Decompose**. The pads are replaced by sampling the encodings directly, and the masks are then computed from the decoding equations rather than from the pads.

The simulator. $\text{Sim}(\text{vk}, \pi, y)$ samples $r_i \leftarrow \mathbb{F}$ and $Z_i \leftarrow \mathbb{G}_2$, sets $U_i = r_i H$, and runs $\text{Encode}^*(\text{vk}, k, \pi, (U_i, Z_i)_i)$ with $k = y$ if π is invalid and with an arbitrary key if π is valid. It outputs the resulting (σ, τ) .

Theorem 1 (Selective security). *If DDH holds in $\mathbb{G}_2$, then the Argo encoding is selectively secure in the standard model with the simulator above. Concretely, for every adversary there is a DDH distinguisher of about the same running time whose advantage is at least the adversary's advantage minus $1/p$.*

Proof. We first record what Encode^* produces in the two cases, for a fixed key k and a proof π chosen before the offline phase.

Real pairs. Let $U_i = r_i H$ and $Z_i = tr_i H$ with $t, r_i \leftarrow \mathbb{F}$. Then $[B]_i = \phi_i B - tU_i$, which is the honest B -encoding. The honest encodings of A, C and x are $\psi_j A - s_j G$, $\phi_i C - v_i G$ and $\phi_i x - w_i$, where the pads s_j, v_i and w_i are uniform, only appear in the masks, and are independent of π . So these encodings are uniform in $\mathbb{G}_1^d \times \mathbb{G}_1^n \times \mathbb{F}^n$ given k, U and $[B]$, and by Lemma 4 the honest masks are the function $p_i = E(\pi)^{\phi_i} \Pi_i^{-1}$ of them. That is exactly how Encode^*

samples the encodings and computes the masks, so its output has the honest distribution.

Random pairs. Let $U_i = r_i H$ and $Z_i \leftarrow \mathbb{G}_2$. Then Z_i is a one-time pad on $\phi_i B$, so $[B]_i$ is uniform and independent of k , and the encodings of A , C and x are sampled without reference to k . The only remaining dependence on k is through $p_i = E(\pi)^{\phi_i} \Pi_i^{-1}$. If π is invalid, the simulator runs Encode^* on the same k and the same distribution of pairs, so its output is identically distributed. If π is valid, $E(\pi) = 1$ and $p_i = \Pi_i^{-1}$ does not involve k , so the output of Encode^* is the same for every key.

Reduction. Given a DDH challenge (P, U, Z) in $\mathbb{G}_2$, the distinguisher

1. Forms n pairs (U_i, Z_i) by Lemma 5
2. Runs $\mathcal{A}_0$ to obtain (k, π)
3. Runs $(\sigma, \tau) = \text{Encode}^*(\text{vk}, k, \pi, (U_i, Z_i)_{i \in [n]})$
4. Outputs $\mathcal{A}_1(\sigma, \tau)$

If the triple is DDH, then by the first observation, $\mathcal{A}_1$ sees the real experiment. If the challenge is random, the pairs are uniform and independent except with probability $1/p$, and by the second observation $\mathcal{A}_1$ sees the ideal experiment. So the distinguisher's advantage is the adversary's advantage minus $1/p$. $\square$

DDH only enters the proof to hide the B -encoding multipliers. The encodings of A , C and x hide the multipliers information-theoretically. The masks p_i hide them because $E(\pi)^{\phi_i}$ is trivial for a valid proof. None of these depend on n , which only enters via the entropy of the key, which the application needs when it commits to the key with a preimage-resistant hash. An adversary that outputs the key on a valid proof with probability ε distinguishes the two cases, since in the ideal case the transcript is independent of k and the output matches with probability $|S|^{-n}$. Thus no adversary outputs the key except with probability $\text{Adv}_{\text{DDH}} + 1/p + |S|^{-n}$.

Remark 1 (Why selective security?). The Encode^* function is only able to produce an encoding when the input is not a DDH triple because it knows the proof. This is because the simulated offline encodings, in particular p_i involve the proof elements. Concretely, an adaptive simulator would have to publish the masks before seeing the proof and then produce an A -encoding consistent with them. This would require tA in $\mathbb{G}_1$ if A is chosen after the masks are fixed. In the DDH reduction, we only have tU_i and do not know t . This appears to be fundamental to this approach, since if they were independent of the proof, then we could use such an encoder to build a version of the protocol that did not rely on DDH. We can't rule out the possibility that a different version of the protocol might be able to avoid the DDH assumption, for example by using different linear combinations of $[A]_j$ for each i , but we leave this to future work. We obtain adaptive security by a black-box transformation instead, in Section 7.2.

Validity Argo. In the variant of Section 6.3 the offline encoding is just $\sigma = U$ and the released value is the mask vector $p = (p_1, \dots, p_n)$. We say the variant is

adaptively unpredictable if no efficient adversary wins the following game except with negligible probability. The challenger first runs the offline phase and gives σ to the adversary. Then the adversary outputs a proof π with $E(\pi) \neq 1$. The challenger gives it $\tau = \text{Online}(\text{st}, \pi)$, and if the adversary outputs $p' = p$ it wins. For a valid proof there is nothing to hide, since the decoder computes p from the transcript.

Proposition 1 (Validity Argo). *If DDH holds in $\mathbb{G}_2$, then Validity Argo is adaptively secure in the standard model and every adversary wins with probability at most $\text{Adv}_{\text{DDH}} + 1/p + |S|^{-n}$.*

Proof. The reduction receives n pairs (U_i, Z_i) as in Lemma 5 and gives $\sigma = U$ to the adversary. It can do this before the proof is known because the offline encoding of this variant doesn't use the masks. When the adversary outputs π , we sample $\phi \leftarrow S^n$ and run $\text{Encode}^*(\text{vk}, \phi, \pi, (U_i, Z_i)_i)$. This yields τ and the masks $p_i = E(\pi)^{\phi_i} \Pi_i^{-1}$. The reduction then gives τ to the adversary, keeps p , and outputs 1 if and only if the adversary outputs p .

If the pairs are real, (σ, τ, p) is distributed as in the game and the B -encoding is honest. The pads s_j , v_i and w_i are uniform and independent of U , and of π , which the adversary chose using only U . Thus the encodings of A , C and x are uniform, and by Lemma 4 the honest masks are $E(\pi)^{\phi_i} \Pi_i^{-1}$, exactly as Encode^* computes them. If the pairs are random, $[B]_i$ is uniform and independent of ϕ along with the rest of the adversary's view. Since $E(\pi) \neq 1$ has order p and distinct elements of S are distinct scalars, the map $\phi \mapsto (E(\pi)^{\phi_i} \Pi_i^{-1})_i$ is injective. Since p is uniform over $|S|^n$ values, the adversary outputs it with probability $|S|^{-n}$. The reduction therefore distinguishes with advantage at least the adversary's winning probability minus $|S|^{-n}$ and $1/p$ from Lemma 5. $\square$

The adaptive proof works here and not for the main scheme because the offline message does not contain the masks, so the reduction never commits to anything that needs to be consistent with the proof. The released vector is unpredictable rather than uniform, since it takes only $|S|^n$ values given the transcript. Hashing it with a function modeled as a random oracle gives a key that is indistinguishable from uniform.

7.2 Adaptive Security

Adaptive security follows in the random oracle model by a standard, black-box transformation [7,30,5]. We just need to encrypt the offline material under the output of the random oracle applied to a uniform key, and reveal the key in the online phase. In Argo, the only parts of the simulated offline encoding that depend on the proof are the masks, so we can encrypt only those and send U_i in the clear.

Concretely, **Offline** additionally samples $\kappa \leftarrow \{0,1\}^\lambda$ and publishes $c = \mathcal{O}(\kappa) \oplus (p_1 \parallel \dots \parallel p_n)$ in place of the masks, where $\mathcal{O}$ is a hash function which we model as a random oracle. **Online** additionally publishes κ , and **Decode** first

recovers the masks as $c \oplus \mathcal{O}(\kappa)$. The transformation adds a single λ -bit string to the online message. It does not change the sizes of the encodings or the work of the decoder beyond one oracle evaluation.

Proposition 2 (Adaptive security). *If DDH holds in $\mathbb{G}_2$ and $\mathcal{O}$ is a programmable random oracle, then the Argo encoding with encrypted masks is adaptively secure. For an adversary making q oracle queries, the advantage bound is that of Theorem 1 plus $q/2^\lambda$.*

Proof. Sim_0 samples $r_i \leftarrow \mathbb{F}$, sets $U_i = r_i H$, samples c uniformly, and outputs $\sigma = (U, c)$. On receiving π and $f(\pi, k)$, Sim_1 runs the selective simulator of Theorem 1 with the same U , obtaining masks p and online encodings τ . It samples $\kappa \leftarrow \{0, 1\}^\lambda$, programs $\mathcal{O}(\kappa) := c \oplus (p_1 \parallel \dots \parallel p_n)$, and outputs (τ, κ) . The programming fails if the adversary successfully guessed κ , which happens with probability at most $q/2^\lambda$ since κ is uniform. Otherwise the view of the adversary is that of the selective experiment with a uniform ciphertext instead of the masks. Theorem 1 applies because the selective simulator's output does not depend on when the proof was chosen. $\square$

8 Projective Argo

The online phase above encodes the proof with an affine function of its elements. In the intended application, the online phase is instead instantiated with the projective Elliptic Curve MAC of Argo MAC [18], or Duty-Free Bits, which produces the same MACs $[A]$, $[B]$, $[C]$ and $[x]$ from an encoding of the bits of the proof. This makes the encoding compatible with tools such as oblivious transfer, at the cost of a garbling table that must be sent in the offline phase. This section shows that we only require privacy from the garbling scheme.

8.1 Garbling the Online Phase

Fix the state st produced by **Offline**. The function being garbled is $\text{Online}(\text{st}, \cdot)$, i.e. the affine map from a proof to its online encoding. A projective garbling scheme for this function consists of three algorithms. $\text{Garble}(\text{st}) \rightarrow (T, e)$ outputs a garbling table T and input-encoding information e . $\text{Encode}(e, \pi) \rightarrow X$ outputs one label for each bit of π . $\text{Eval}(T, X) \rightarrow \tau$ recovers the online encoding. Correctness requires that $\text{Eval}(T, \text{Encode}(e, \pi))$ equals $\text{Online}(\text{st}, \pi)$ for every π .

The composed scheme publishes (U, p, T) in the offline phase. In the online phase it publishes X and the decoder runs **Eval** and then **Decode**. We need two properties of the garbling scheme [8].

- **Privacy.** There exists a simulator Sim_G such that for every st and every π fixed in advance, (T, X) is computationally indistinguishable from $\text{Sim}_G(\tau)$ where $\tau = \text{Online}(\text{st}, \pi)$. The simulator does not receive the private MAC keys.

- **Authenticity.** No efficient adversary can choose π such that, given (T, X) , it outputs $\text{Online}(\text{st}, \pi')$ for a proof $\pi' \neq \pi$ except with negligible probability.

This is the privacy notion of Section 3.5 where the public input is the proof and the private inputs are the MAC keys. It is exactly Definition 1 of Duty-Free Bits [37], the simulator receives the evaluator's input and the output but not the key. Argo MAC [18] states its security as authenticity, but its proof in Section 3.5 is a simulation argument that outputs the table and the input encoding without the garbling key. This is the essentially the same property, so both instantiations work.

The papers also guarantee correctness and privacy only if the encoded input is a valid curve point. Nothing stops a cheating prover from publishing labels for bit strings that do not encode a point. This would cause the projective garbling schemes to fail and break the overall protocol. So we require that the projective MAC include a validity check that leaks a secret on invalid curve points. Argo MAC includes this check, and the secret it leaks can be made the key itself. We assume the projective MAC scheme we use includes such a check and any other checks necessary to ensure the proof is syntactically valid.

In general, privacy and authenticity are independent properties garbling schemes, but for our function they are not. This is because a forgery of the affine MAC reveals the multipliers, which is the argument of Lemma 2 extended across the pairing.

Lemma 6 (A forgery reveals the multipliers). *Let $\pi = (A, B, C, x)$ with $B \neq 0$, and let $\tau = \text{Online}(\text{st}, \pi)$. Given τ , π , the offline encoding σ , and $\tau' = \text{Online}(\text{st}, \pi')$ for any $\pi' = (A', B', C', x') \neq \pi$, we can compute ϕ .*

Proof. If π' is invalid, run Decode on (σ, τ', π') and we immediately get ϕ . Otherwise π' is valid and we can cancel the masks from at least one of the encoded values. For x , we can divide out the difference and get ϕ . For B, C we can solve a small discrete log problem. For A , we can compute $[A]' - [A] = t(A' - A)$, which does not immediately expose the multipliers. Pairing it with U_i and using $tU_i = \phi_i B - [B]_i$ gives

$$e([A]' - [A], U_i) = e(A' - A, tU_i) = e(A' - A, B)^{\phi_i} e(A' - A, [B]_i)^{-1},$$

so $e(A' - A, B)^{\phi_i}$ is computable from the two transcripts, which is again a small discrete log problem. Therefore, we can extract ϕ . $\square$

Lemma 7 (Privacy implies authenticity for the online phase). *If the garbling scheme is private and DDH holds in $\mathbb{G}_2$, then the garbling scheme is authentic.*

Proof. Consider a forger $\mathcal{A}$ that succeeds with probability ε on proofs with $B \neq 0$. Run $\mathcal{A}$ on $\text{Sim}_G(\tau)$ instead of (T, X) , and let ε' be its success probability. A forgery is efficiently checkable given st , so the difference $|\varepsilon - \varepsilon'|$ is the distinguisher advantage. A forger that succeeds on $\text{Sim}_G(\tau)$ computes $\text{Online}(\text{st}, \pi')$ from σ , τ and π , and then by Lemma 6 computes the key ϕ . This contradicts theorem 1

for a valid proof, as it shows the probability of outputting the key from (σ, τ) by the DDH advantage plus $1/p + |S|^{-n}$. For an invalid π the decoder already outputs ϕ . So ε is bounded by the privacy advantage plus that quantity. $\square$

Here we assume the adversary holds labels for a single proof, since the encoder is incentivized to release labels only for a valid proof. Given multiple labels, the decoder could construct an encoding of an invalid proof. If this was somehow not the case, the overall protocol would still be secure since the encoder has only published valid proofs.

8.2 Security of the Composed Scheme

Theorem 2 (Selective security of the composed scheme). *If the garbling scheme is private and the Argo encoding is selectively secure, then the composed scheme is selectively secure.*

Proof. The composed simulator takes π and $f(\pi, k)$ and runs the Argo simulator of Theorem 1 to obtain (σ, τ) . Then it runs $\text{Sim}_G(\tau)$ to obtain (T, X) and outputs (σ, T) as the offline encoding and X as the online encoding. Two hybrids connect this to the real experiment.

First, replace the real (T, X) by $\text{Sim}_G(\tau)$ for the real τ . In the privacy reduction we sample the Argo state itself, compute σ and τ honestly, and submit st and π to the privacy challenger. Privacy holds over every function and input, so the correlation between the adversary learns nothing from the garbled function and the offline encoding σ .

Second, the view is now $(\sigma, \text{Sim}_G(\tau))$ which is a function of the Argo transcript. Theorem 1 states (σ, τ) is indistinguishable from the Argo simulator's output. Applying Sim_G to τ preserves indistinguishability. $\square$

We don't use authenticity anywhere. Note that an evaluator that holds labels for a valid proof and manufactures the online encoding of an invalid one would recover k , but Theorem 2 rules this out, so the forgery is excluded without a separate argument. This explicit at the level of the garbling scheme in Lemma 7.

8.3 Adaptive Security

The garbling table depends on the MAC keys, so adaptive security does not immediately follow from the adaptive security of the encoding. Duty-Free Bits [37] also has this issue, and we resolve it the same way. The encoder also encrypts the garbling table under $\mathcal{O}(\kappa)$ and publishes the ciphertext in the offline phase. In the online phase it publishes κ with the labels. The proof of Proposition 2 is the mostly the same with the table included in the encrypted material and the with the composed simulator in place of the Argo simulator. This shows the composed scheme is adaptively secure in the random oracle model if the garbling scheme is private.

Acknowledgements. We used Claude Fable for L^AT_EX formatting, review of the manuscript, and help exposition for clarity. The protocol, the security proof and the implementation were developed independently by the authors.

References

1. Applebaum, B.: Garbled circuits as randomized encodings of functions: a primer. Cryptology ePrint Archive, Paper 2017/385 (2017), <https://eprint.iacr.org/2017/385>
2. Applebaum, B., Ishai, Y., Kushilevitz, E.: How to garble arithmetic circuits. In: 52nd FOCS. pp. 120–129. IEEE Computer Society Press (2011)
3. Ball, M., Li, H., Lin, H., Liu, T.: New ways to garble arithmetic circuits. In: EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 3–34. Springer (2023)
4. Ball, M., Malkin, T., Rosulek, M.: Garbling gadgets for Boolean and arithmetic circuits. In: ACM CCS 2016. pp. 565–577. ACM (2016)
5. Barnum, C., Heath, D., Kolesnikov, V., Ostrovsky, R.: Adaptive garbled circuits and garbled RAM from non-programmable random oracles. Cryptology ePrint Archive, Paper 2023/1527 (2023), <https://eprint.iacr.org/2023/1527>
6. Bartusek, J., Ishai, Y., Jain, A., Ma, F., Sahai, A., Zhandry, M.: Affine determinant programs: A framework for obfuscation and witness encryption. In: ITCS 2020. LIPIcs, vol. 151, pp. 82:1–82:39. Schloss Dagstuhl (2020)
7. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: ASIACRYPT 2012. LNCS, vol. 7658, pp. 134–153. Springer (2012)
8. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: ACM CCS 2012. pp. 784–796. ACM (2012)
9. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer (2011)
10. BitVM Alliance: garbled-snark-verifier. <https://github.com/BitVM/garbled-snark-verifier> (2025)
11. Boneh, D.: The decision Diffie-Hellman problem. In: ANTS-III. LNCS, vol. 1423, pp. 48–63. Springer (1998)
12. Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., Vinayagamurthy, D.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: EUROCRYPT 2014. LNCS, vol. 8441, pp. 533–556. Springer (2014)
13. Cho, C., Döttling, N., Garg, S., Gupta, D., Miao, P., Polychroniadou, A.: Laconic oblivious transfer and its applications. In: CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 33–65. Springer (2017)
14. Cramer, R., Fehr, S., Ishai, Y., Kushilevitz, E.: Efficient multi-party computation over rings. In: EUROCRYPT 2003. LNCS, vol. 2656, pp. 596–613. Springer (2003)
15. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer (2002)
16. Döttling, N., Garg, S., Ishai, Y., Malavolta, G., Mour, T., Ostrovsky, R.: Trapdoor hash functions and their applications. In: CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 3–32. Springer (2019)

17. Eagen, L.: Glock: Garbled locks for Bitcoin. Cryptology ePrint Archive, Paper 2025/1485 (2025), <https://eprint.iacr.org/2025/1485>
18. Eagen, L., Lai, Y.T.: Argo MAC: Garbling with elliptic curve MACs. Cryptology ePrint Archive, Paper 2026/049 (2026), <https://eprint.iacr.org/2026/049>
19. Eldridge, H., Goel, A., Green, M., Jain, A., Zinkus, M.: One-time programs from commodity hardware. In: TCC 2022, Part III. LNCS, vol. 13749, pp. 121–150. Springer (2022), cryptology ePrint Archive, Paper 2022/1257
20. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: CRYPTO '84. LNCS, vol. 196, pp. 10–18. Springer (1985)
21. Fleischhacker, N., Hall-Andersen, M., Simkin, M.: Extractable witness encryption for KZG commitments and efficient laconic OT. In: ASIACRYPT 2024, Part II. LNCS, vol. 15485, pp. 423–453. Springer (2024)
22. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16), 3113–3121 (2008)
23. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer (2001)
24. Garg, S., Hajiabadi, M., Kolonelos, D., Kothapalli, A., Policharla, G.V.: A framework for witness encryption from linearly verifiable SNARKs and applications. Cryptology ePrint Archive, Paper 2025/1364 (2025), <https://eprint.iacr.org/2025/1364>
25. Garg, S., Kolonelos, D., Policharla, G.V., Wang, M.: Threshold encryption with silent setup. In: CRYPTO 2024, Part VII. LNCS, vol. 14926, pp. 352–386. Springer (2024)
26. Garg, S., Kolonelos, D., Sergeevitch, M., Sridhar, S., Tse, D.: BABE: Verifying proofs on Bitcoin made 1000x cheaper. Cryptology ePrint Archive, Paper 2026/065 (2026), <https://eprint.iacr.org/2026/065>
27. Gebali, M.E., Rebenko, Y., Schofnegger, M., Soukhanov, L.: Breaking ADP-based witness encryption. Cryptology ePrint Archive, Paper 2026/1594 (2026), <https://eprint.iacr.org/2026/1594>
28. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer (2016)
29. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer (2008)
30. Guo, X., Yang, K., Wang, X., Yu, Y., Liu, Z.: Unmodified half-gates is adaptively secure – so is unmodified three-halves. Cryptology ePrint Archive, Paper 2023/1528 (2023), <https://eprint.iacr.org/2023/1528>
31. Heath, D.: Efficient arithmetic in garbled circuits. In: EUROCRYPT 2024, Part V. LNCS, vol. 14655, pp. 3–31. Springer (2024)
32. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: 41st FOCS. pp. 294–304. IEEE Computer Society Press (2000)
33. Ishai, Y., Kushilevitz, E.: Perfect constant-round secure computation via perfect randomizing polynomials. In: ICALP 2002. LNCS, vol. 2380, pp. 244–256. Springer (2002)
34. Ishai, Y., Li, H., Lin, H.: Succinct homomorphic MACs from groups and applications. Cryptology ePrint Archive, Paper 2024/2073 (2024), <https://eprint.iacr.org/2024/2073>, earlier title: Succinct Partial Garbling from Groups and Applications
35. Ishai, Y., Wee, H.: Partial garbling schemes and their applications. In: ICALP 2014, Part I. LNCS, vol. 8572, pp. 650–662. Springer (2014)

36. Jo, S.: Cryptanalysis of a candidate witness encryption scheme for affine determinant programs. Cryptology ePrint Archive, Paper 2026/1583 (2026), <https://eprint.iacr.org/2026/1583>
37. Khambhati, N., Bhattacharya, A., Heath, D.: Duty-free bits: Projectivizing garbling schemes. Cryptology ePrint Archive, Paper 2026/476 (2026), <https://eprint.iacr.org/2026/476>
38. Khambhati, N., Tiwari, M., Azz, Bajracharya, S., Bista, M., Eagen, L., Lewe, C., Feickert, A.: Mosaic: Practical malicious security for garbled circuits on Bitcoin. Cryptology ePrint Archive, Paper 2026/812 (2026), <https://eprint.iacr.org/2026/812>
39. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer (2008)
40. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer (2013)
41. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer (2007)
42. Linus, R.: BitVM: Compute anything on Bitcoin. <https://bitvm.org/bitvm.pdf> (2023)
43. Linus, R., Aumayr, L., Avarikioti, Z., Maffei, M., Pelosi, A., Litos, O.T., Stefo, C., Tse, D., Zamyatin, A.: Bridging Bitcoin to second layers via BitVM2. Cryptology ePrint Archive, Paper 2025/1158 (2025), <https://eprint.iacr.org/2025/1158>
44. Rubin, J.: Delbrag. <https://rubin.io/public/pdfs/delbrag.pdf> (2025)
45. Soukhanov, L., Rebenko, Y., Gebali, M.E., Komarov, M., Kilinc-Alper, H., Abdalla, M., Towa, P.: Implementable witness encryption from arithmetic affine determinant programs. Cryptology ePrint Archive, Paper 2026/175 (2026), <https://eprint.iacr.org/2026/175>
46. Thibault, L.T., Sarry, T., Hafid, A.S.: Blockchain scaling using rollups: A comprehensive survey. IEEE Access **10**, 93039–93054 (2022)
47. Wegman, M.N., Carter, J.L.: New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences **22**(3), 265–279 (1981)
48. Woll, R.L., Alexopoulos, I., Aumayr, L., Avarikioti, Z., Maffei, M., Tse, D.: BitVM3: Efficient Bitcoin bridges via garbled circuits. Cryptology ePrint Archive, Paper 2026/933 (2026), <https://eprint.iacr.org/2026/933>
49. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Yellow Paper, <https://ethereum.github.io/yellowpaper/paper.pdf> (2014)
50. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In: EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 220–250. Springer (2015)