

CRYPTOGRAPHY · BILINEAR MAPS · NIZK

GSkit

A domain-specific language and framework for Groth-Sahai proofs

Benjamín Hurtado Torrealba

Advisors: Matías Toro Ipinza · Federico Olmedo Berón

Departamento de Ciencias de la Computación · Universidad de Chile

Magíster en Ciencias, mención Computación · Ingeniería Civil en Computación

TUESDAY 13 OCTOBER 2026

benjamin.hurta.do/gskit

Agenda

1 Zero-knowledge proofs

2 Signatures & bilinear maps

3 Groth-Sahai proofs

4 GSkit: the DSL and framework

5 Case study: ABS-UCL

6 Evaluation & conclusions

Why prove without revealing?

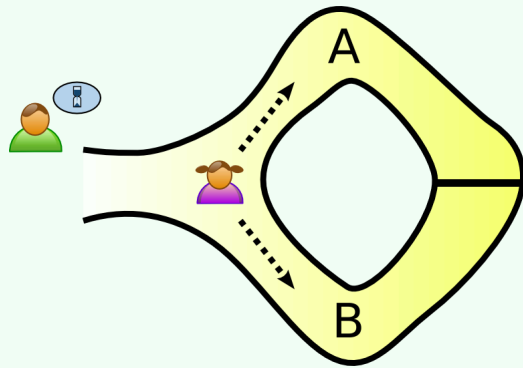
Every day we have to **convince someone of a fact**, and usually we do it by handing over the evidence: an ID to prove our age, a password to prove it's us.

But the evidence is often a secret we'd rather not give away. Can we prove the statement is true **while keeping the secret to ourselves**?

That is the question zero-knowledge proofs answer.

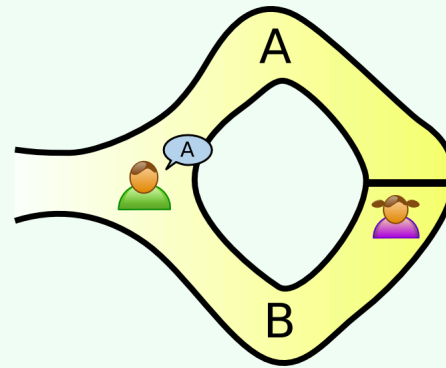
Zero-knowledge, by example

Peggy (the *prover*) claims she knows the secret word that opens the door inside Ali Baba's cave. Victor (the *verifier*) wants to be convinced, without learning the word.



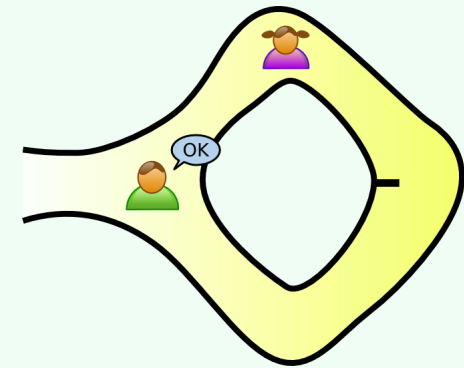
1 · Commit

Peggy walks in through A or B at random, while Victor waits outside



2 · Challenge

Victor shouts the side she must come out from



3 · Response

Peggy comes out on that side, through the door if she has to

Three properties

Repeating the cave game makes the three properties of a zero-knowledge proof concrete:

Completeness

An honest prover with a valid witness always convinces the verifier

If Peggy knows the word, she always comes out where Victor asks

Soundness

A cheating prover cannot convince the verifier of a false statement

Without the word she gets caught half of the time: after k rounds she fools Victor with probability 2^{-k}

Zero-knowledge

The verifier learns nothing beyond the truth of the statement

Victor only sees her come out where he asked, a scene he could have staged himself

Formally — NIZK

Consider an NP-relation $\mathcal{R}$ and the language it defines:

$$\mathcal{L} := \{ x \mid \exists w \text{ s.t. } (x, w) \in \mathcal{R} \}$$

where x is the **statement** (public) and w the **witness** (secret). A *non-interactive* zero-knowledge proof (NIZK) is a single message π , checked against a common reference string:

Setup — $\text{CRS} \leftarrow \text{Setup}(1^\lambda)$

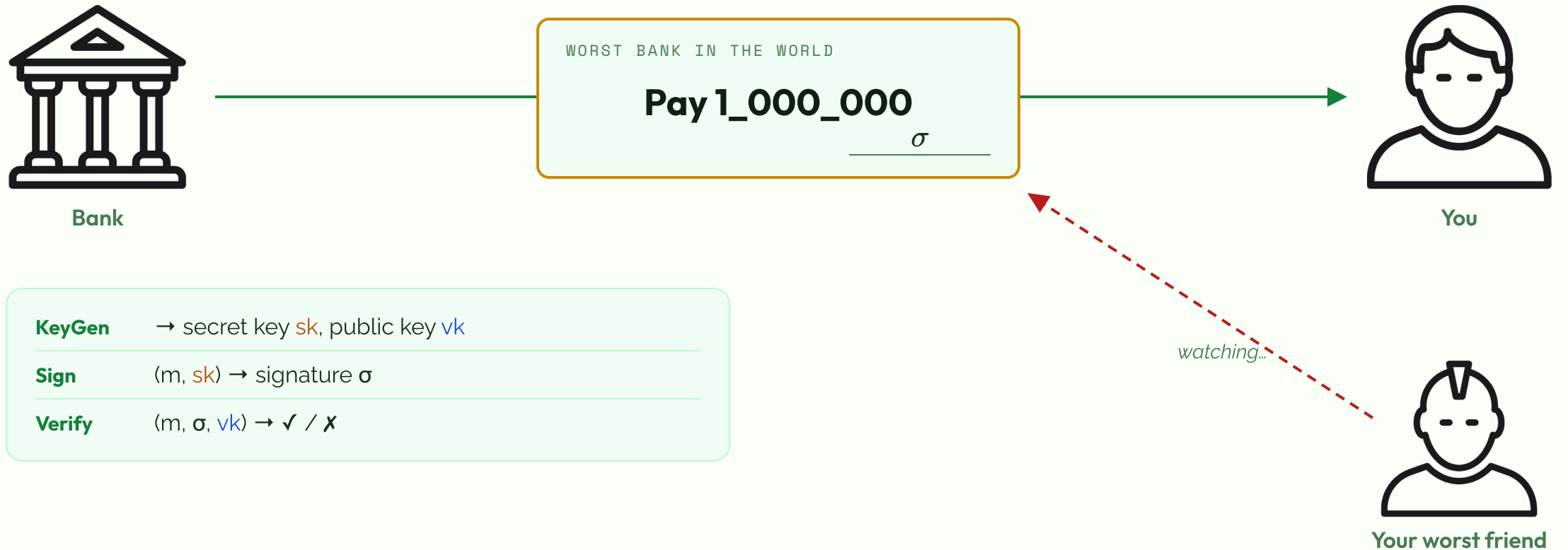
Prove — $\pi \leftarrow \text{Prove}(\text{CRS}, x, w)$

Verify — $b \leftarrow \text{Verify}(\text{CRS}, x, \pi)$

● secret values: the same color marks them on every slide

Worst Bank in the World

To make this concrete, meet the **worst bank in the world**. It issues cheques, and everyone around it needs a way to tell a genuine one from a forgery.



Signatures, in one idea

A **signature scheme** is the bank's anti-forgery tool. The bank keeps a secret key sk and publishes a verification key vk : anyone can check, nobody can forge.

- **KeyGen** — produces the key pair (sk, vk) . The secret stays with the bank; vk goes public.
- **Sign** — only the holder of sk can turn a message m into a signature σ .
- **Verify** — anyone with vk can confirm that σ really matches m .

The ingredient that makes a **good** signature scheme possible is the **bilinear map**.

Bilinear maps

A **bilinear map** is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ that lets us **move scalars across a pairing**:

$$e(a g_1, b g_2) = e(g_1, g_2)^{ab}$$

- $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are finite groups of prime order p , defined over elliptic curves
- g_1, g_2 generate $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $a, b \in \mathbb{Z}_p$

This one property, pulling exponents out of a pairing, is the lever behind both BLS signatures and Groth-Sahai proofs.

BLS signature

Boneh–Lynn–Shacham [BLS04] turn that lever into a signature scheme: the secret key hides as an **exponent**, and the pairing lets anyone check it **without ever seeing it**. A hash-to-curve $\mathcal{H} : \{0, 1\}^* \rightarrow \mathbb{G}_1$ maps messages onto the curve.

$$\textbf{KeyGen} \quad - \quad sk \xleftarrow{R} \mathbb{Z}_p \quad vk \leftarrow [sk] g_2$$

$$\textbf{Sign} \quad - \quad h \leftarrow \mathcal{H}(m) \quad \sigma \leftarrow [sk] h$$

$$\textbf{Verify} \quad - \quad h \leftarrow \mathcal{H}(m) \quad e(\sigma, g_2) \stackrel{?}{=} e(h, vk)$$

Verification works because bilinearity makes both sides collapse to the same value:

$$e([sk]h, g_2) = e(h, g_2)^{sk} = e(h, [sk]g_2)$$

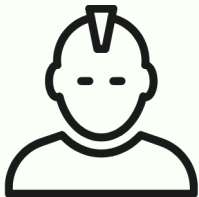
Worst Bank in the World uses BLS

So the bank adopts BLS: it writes a cheque and **signs** it with its secret key. Anyone can verify it; only the bank could have produced it.



Pay 1_000_000

σ



Your worst friend

watching...

The bank **signs** the cheque using BLS:

$$\sigma \leftarrow [sk] \mathcal{H}(\text{"Pay 1_000_000"})$$

Anyone with the verification key vk can verify it:

$$e(\sigma, g_2) \stackrel{?}{=} e(\mathcal{H}(\text{"Pay 1_000_000"}), vk)$$

Just by **seeing** σ , your worst friend can **cash the cheque**: the bank pays whoever shows a valid signature.

A new problem

The cheque is signed and verifiable. Now **you hold a valid cheque** and want to prove to a friend that *the bank really signed it*, without handing over σ , or they could cash it themselves.

You want to prove "*a valid σ exists*", i.e. that the verification equation **has a solution**, while keeping σ hidden.

That's exactly a **zero-knowledge proof about a bilinear-map equation**. Enter Groth-Sahai.

Groth-Sahai proofs

Groth-Sahai proofs [GS08] are efficient **non-interactive zero-knowledge proofs** for statements written as **equations over a bilinear group**. The richest kind is the *pairing-product equation* (PPE):

$$\prod_{i=1}^n e(\mathcal{A}_i, \mathcal{Y}_i) \cdot \prod_{i=1}^m e(\mathcal{X}_i, \mathcal{B}_i) \cdot \prod_{i=1}^m \prod_{j=1}^n e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{ij}} = t_T$$

CONSTANTS · PUBLIC

$$\mathcal{A}_i \in \mathbb{G}_1, \mathcal{B}_i \in \mathbb{G}_2, t_T \in \mathbb{G}_T, \gamma_{ij} \in \mathbb{Z}_p$$

WITNESS · SECRET

$$\mathcal{X}_1, \dots, \mathcal{X}_m \in \mathbb{G}_1, \mathcal{Y}_1, \dots, \mathcal{Y}_n \in \mathbb{G}_2$$

The prover knows a witness that satisfies the equation; the verifier becomes convinced **without ever learning it**.

Four equation types

The same quadratic shape, over a different bilinear map, gives **four equation types**:

PPE · PAIRING PRODUCT

$$\prod_i e(\mathcal{A}_i, \mathcal{Y}_i) \cdot \prod_i e(\mathcal{X}_i, \mathcal{B}_i) \cdot \prod_{i,j} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{ij}} = t_T$$

Witness: $\mathcal{X}_i \in \mathbb{G}_1, \mathcal{Y}_j \in \mathbb{G}_2$

MSME · MULTI-SCALAR IN $\mathbb{G}_1$

$$\sum_i y_i \mathcal{A}_i + \sum_i b_i \mathcal{X}_i + \sum_{i,j} \gamma_{ij} y_j \mathcal{X}_i = \mathcal{T}_1$$

Witness: $\mathcal{X}_i \in \mathbb{G}_1, y_j \in \mathbb{Z}_p$

MSME · MULTI-SCALAR IN $\mathbb{G}_2$

$$\sum_i a_i \mathcal{Y}_i + \sum_i x_i \mathcal{B}_i + \sum_{i,j} \gamma_{ij} x_i \mathcal{Y}_j = \mathcal{T}_2$$

Witness: $x_i \in \mathbb{Z}_p, \mathcal{Y}_j \in \mathbb{G}_2$

QE · QUADRATIC IN $\mathbb{Z}_p$

$$\sum_i a_i y_i + \sum_i x_i b_i + \sum_{i,j} \gamma_{ij} x_i y_j \equiv t \pmod{p}$$

Witness: $x_i, y_j \in \mathbb{Z}_p$

Formally — the Groth-Sahai language

$$\mathcal{L} := \left\{ \text{statement} \mid \exists \text{witness} : \mathcal{E}_1(\text{stmt}, \text{wit}), \dots, \mathcal{E}_n(\text{stmt}, \text{wit}) \text{ hold} \right\}$$

where each $\mathcal{E}_i$ is one of the **four Groth-Sahai equation types** over the variables:

Group variables

$$\mathcal{X}_1, \dots, \mathcal{X}_m \in \mathbb{G}_1$$

$$\mathcal{Y}_1, \dots, \mathcal{Y}_n \in \mathbb{G}_2$$

Scalar variables

$$\mathcal{X}_1, \dots, \mathcal{X}_{m'} \in \mathbb{Z}_p$$

$$\mathcal{Y}_1, \dots, \mathcal{Y}_{n'} \in \mathbb{Z}_p$$

One proof shows that *all* the equations hold for one shared witness. GSKit's DSL, coming next, is a surface syntax for exactly this language: declare the witness and the equations, and the framework builds the proof.

The problem we want to solve

Writing one of these proofs **by hand** is intricate and error-prone: you commit every variable, build the a , b vectors and the γ matrix, and keep the prover's and verifier's views perfectly in sync.

The pain

Low-level commitments, easy to get wrong, hard to read; prover and verifier can drift apart.

The idea

Describe *what* the equation is, declaratively, and let a compiler generate the rest.

So we designed a small **DSL**: you declare the equation once, and the framework compiles both proving and verification.

Related work — and the gap

Zero-knowledge DSLs

Circom · ZoKrates · Noir · Cairo · Leo

They compile programs to **circuits** (R1CS, ACIR, AIR) for SNARK and STARK back-ends. None supports bilinear-group equations natively.

Groth-Sahai libraries

groth-sahai-rs · groth-sahai-python · a Java prototype

Programmatic APIs: the developer still classifies witnesses, picks equation types, builds commitments and names variables across sub-proofs.

GSkit fills the gap: Groth-Sahai statements written as declarative **equations**, not as circuits or API calls.

This thesis

- 1** **A DSL for Groth-Sahai statements.** Declare variables, constants and equations; a type system infers each equation's type (PPE, MSME, QE) and rejects ill-formed specs *before* any proof is built.

- 2** **GSkit, a Python framework** on Charm/PBC that compiles one spec into **both** the prover and the verifier, from the command line or from Python code with `@gsfy`.

- 3** **Two case studies**, BLS and ABS-UCL, the latter compared against an earlier hand-written implementation.

A DSL for Groth-Sahai

A specification has just **three blocks**: which names are **hidden** (variables), which are **public** (constants), and the **equations**:

variables:

```
<hidden names> : G1 | G2 | ZP
```

constants:

```
<public names> : G1 | G2 | ZP | GT
```

equations:

```
<term> + <term> + ... = <name>
```

variables — the witness, what stays secret (only `G1`, `G2`, `ZP`)

constants — public values, including the right-hand side (`GT` allowed here)

equations — sums of product terms; `*` is the bilinear operation, `+` combines terms

Reading a term

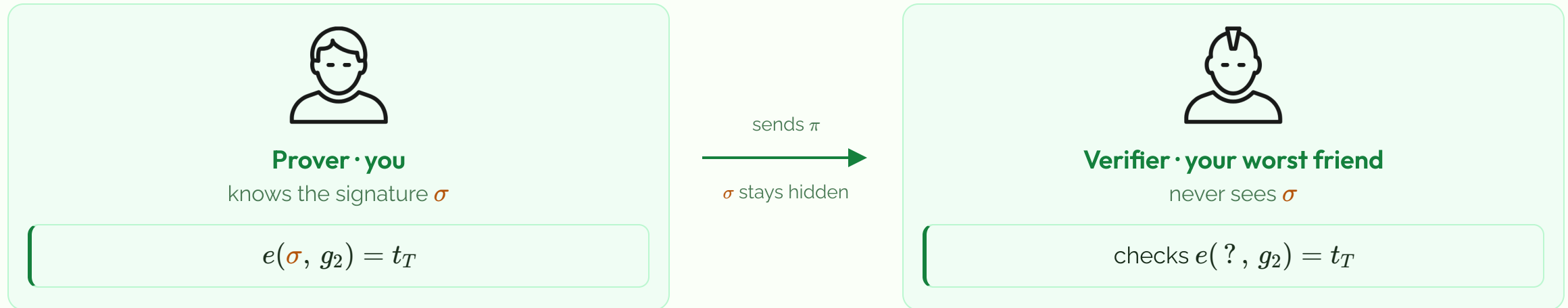
Each term is a product of names. **Which groups the names live in** decides the equation type, and the compiler infers it for you:

$$\prod_{i=1}^n e(\mathcal{A}_i, \mathcal{Y}_i) \cdot \prod_{i=1}^m e(\mathcal{X}_i, \mathcal{B}_i) \cdot \prod_{i,j} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{ij}} = t_T$$

- **CONST · VAR** $e(\mathcal{A}_i, \mathcal{Y}_i) \rightarrow$ written **A * Y**
- **VAR · CONST** $e(\mathcal{X}_i, \mathcal{B}_i) \rightarrow$ written **X * B**
- **VAR · VAR, SCALED** $e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{ij}} \rightarrow$ written **gamma * X * Y**

BLS proof to my worst friend

Back to the cheque. **You** (the prover) know σ ; your friend (the verifier) must be convinced without seeing it.



Key insight: your friend learns that *a valid signature exists*, but never sees σ itself.

$$t_T \leftarrow e(\mathcal{H}(\text{"Pay 1_000_000"}), vk) \quad e(\sigma, g_2) = t_T$$

The BLS proof, in the DSL

The whole statement "*I know a σ such that $e(\sigma, g_2) = t_T$* " becomes a few lines. The secret σ is a variable; everything else is a constant:

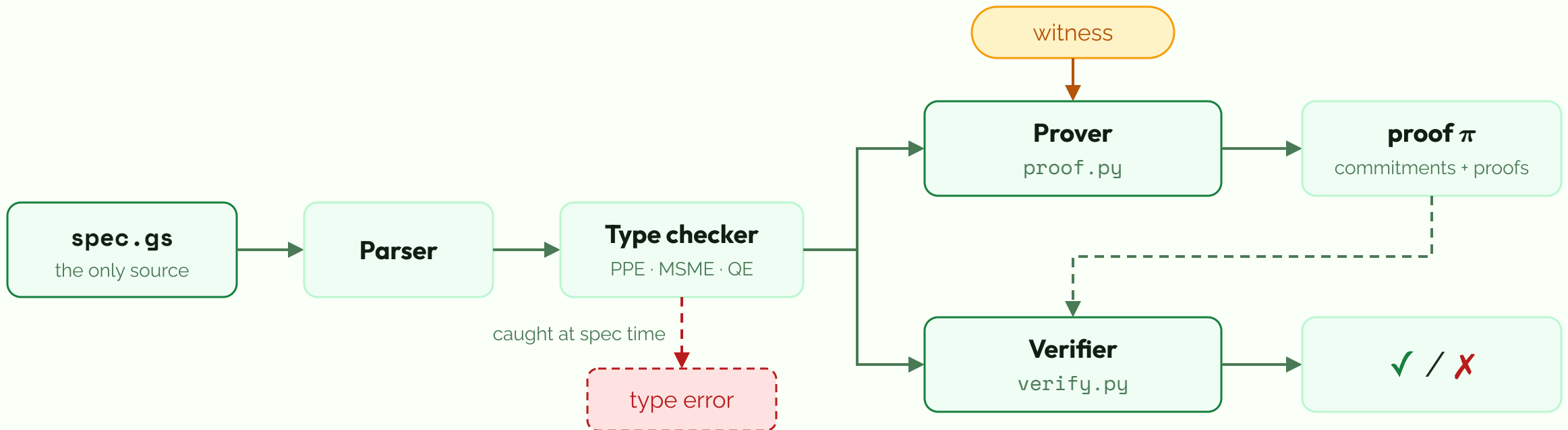
- σ is the **witness** → declared under `variables`
- g_2 and the target t_T are **public** → `constants`
- a single **PPE**: `signature * g2 = rhs`

The compiler reads the types and recognises a pairing-product equation, with no manual setup.

```
variables:  
  signature: G1  
  
constants:  
  g2: G2  
  rhs: GT  
  
equations:  
  signature * g2 = rhs
```

One spec, both sides

The same typed spec compiles into **two** programs, so the prover's and the verifier's view of the equations can never drift apart:



Inside Python, the `@gsfy` decorator reads the same spec from a `GS_STRING` in the function and wires its variables automatically.

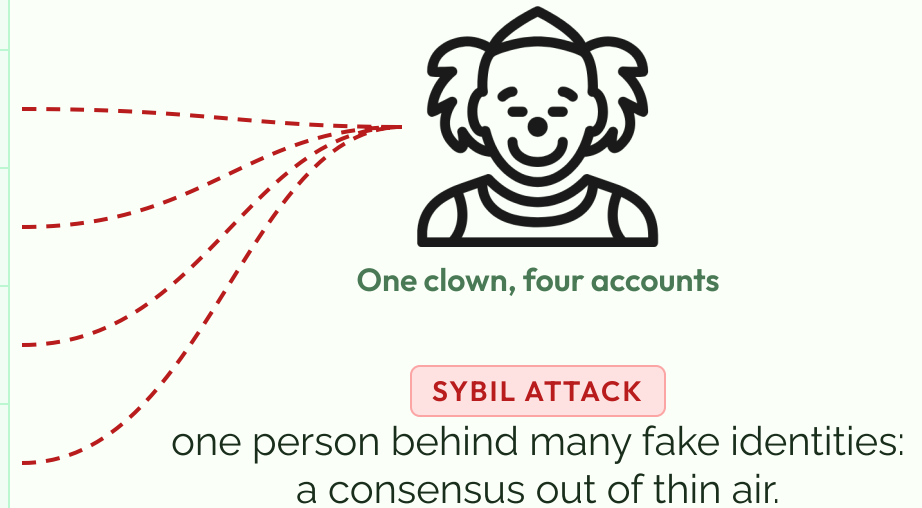
Where it all began: an anonymous forum

BLS was a warm-up. This thesis started from a concrete goal: an **anonymous forum**, where members prove they belong without revealing who they are. But anonymity alone has a catch:

ANONYMOUS FORUM • #PARK-POLICY 4 POSTS

	anon_4821	In my opinion we should kick puppies at the park.
	anon_1337	Totally agree. Finally someone said it.
	anon_0042	+1. Those puppies had it coming.
	dog_lover_77	As a dog lover, I fully support this.

▲ 4 agree • 0 disagree



Not just a forum problem: sockpuppet accounts and bot farms manufacture consensus in today's public debate and elections, a real threat to democracy.

The fix: ABS-UCL

Attribute-based signatures with user-controlled linkability [EKCGD14, EG17]. Every post carries one signature proving:

- "my attributes satisfy the policy" (say, *UChile student*), revealing neither who signed nor which attributes;
- a well-formed **linking token** τ , derived from the author's secret key *usk* and the thread.

#PARK-POLICY

SIGNED WITH ABS-UCL

τ 9c4e...a1

In my opinion we should kick puppies at the park.

τ 9c4e...a1

Totally agree. Finally someone said it.

τ 9c4e...a1

+1. Those puppies had it coming.

τ 9c4e...a1

As a dog lover, I fully support this.

4 posts · 1 author

Same author, same thread, same τ : the four "supporters" count as **one voice**. Across threads the tokens look unrelated, so authors stay anonymous. Privacy **and** accountability, in one signature.

One signature, four sub-protocols

A single ABS-UCL signature is a **conjunction** of zero-knowledge statements, each from an independent sub-protocol: many Groth-Sahai equations proved as one.

RPSPS

The attribute keys the user holds were **genuinely issued** by the authorities.

Re-randomizable structure-preserving signatures

MSP

Those attributes **satisfy the access policy**.

Monotone span program

LIT

The **linking token** is well-formed: the key to controlled linkability.

Linking token

FBB • PSDO

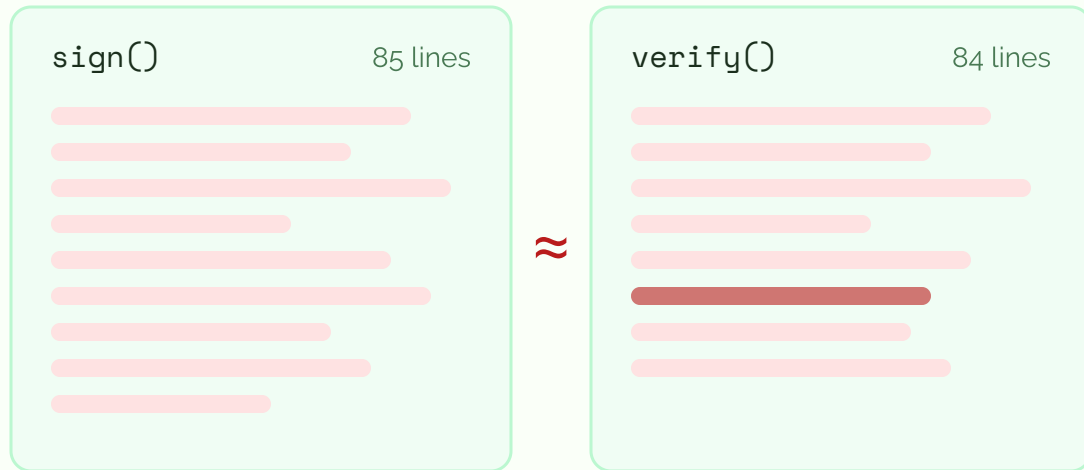
A **pseudo-attribute** provides a distinguished branch, so the policy is always satisfiable.

Full Boneh–Boyen signature [BB04]

The payoff: compose, don't duplicate

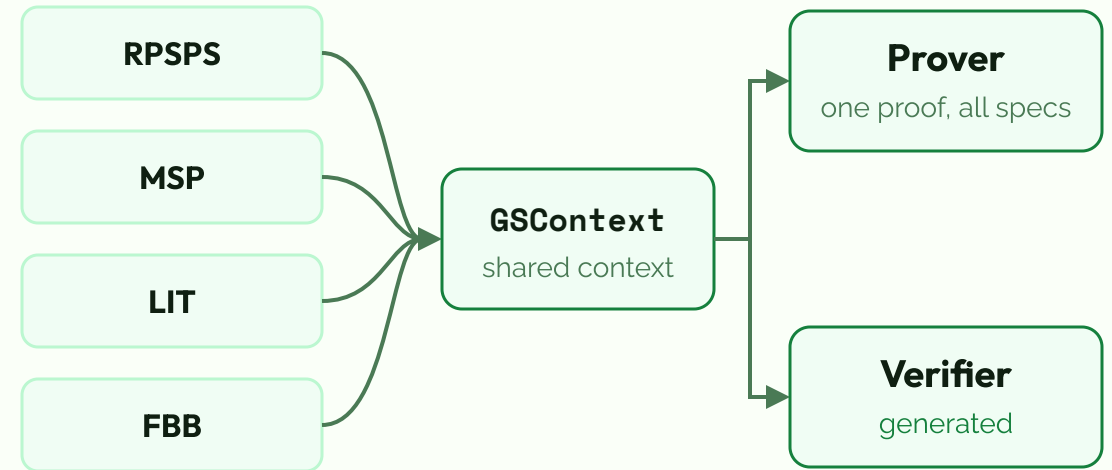
Each sub-protocol declares its Groth-Sahai logic **once** with `agsfy`; the scheme composes these specs over a shared context, and both prover and verifier come from that single source.

The old way • manual



every equation is written twice, and the copies can drift apart

GSkit • modular



each equation is written once, inside its sub-protocol

The result

The decisive evidence is not fewer total lines, but the **elimination of duplicated, hand-maintained equation code**:

Equation-construction effort	Manual	GSkit (modular)
Lines in <code>sign()</code>	85	8 declarative equations
Lines in <code>verify()</code>	84	0, generated
Hand-built equation / pairing-map objects	60	0
Times each equation is written	2	1

Total developer-written code is not smaller (420 → 540 lines): each sub-protocol now carries its own spec.

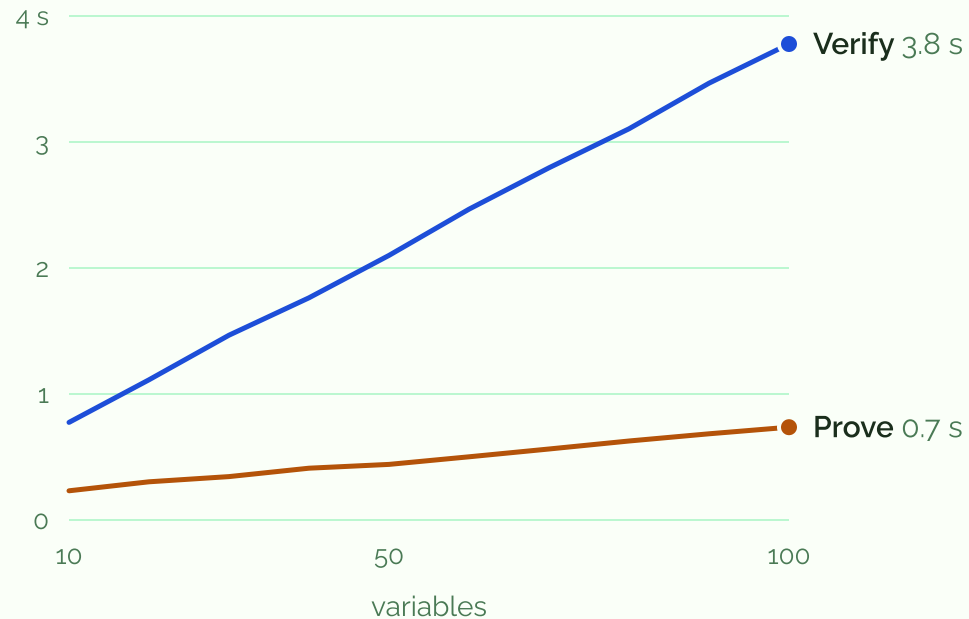
The contribution: moving the equations into a single declarative source is what lowers the error surface, and makes a scheme as complex as ABS-UCL practical to build and maintain.

Performance

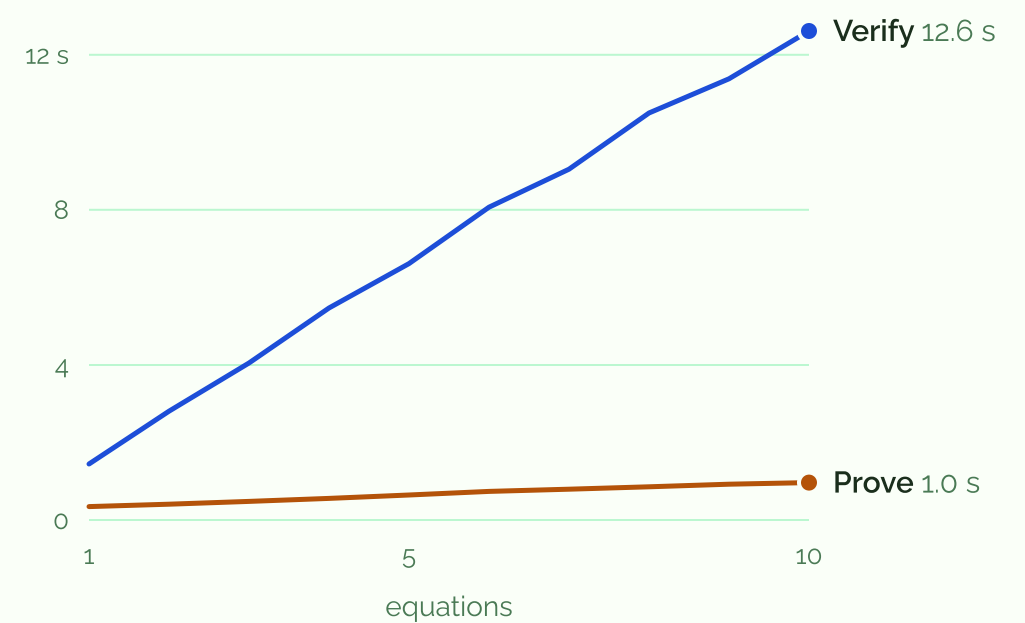
Proving and verifying a pairing-product statement takes time **linear** in the number of variables and in the number of equations.

— Verify — Prove

Growing the number of variables



Growing the number of equations



GSkit on Charm/PBC, BN254 curve. Data from the thesis evaluation.

Conclusions

What we showed

- One declarative spec per statement: prover **and** verifier are generated from it, so they cannot drift apart.
- Type checking moves a class of errors from run time to **specification time**.
- On ABS-UCL, 169 hand-written lines and 60 equation objects become **8 declarative equations**, written once.

Future work

- Batch verification and proof-size optimizations
- More case studies beyond BLS and ABS-UCL
- End to end: the anonymous attribute-based forum that motivated this work

Hypothesis confirmed: a domain-specific language lowers the barrier to building Groth-Sahai proofs, reducing effort and error surface without giving up their security guarantees.

Thank you

Questions?



Benjamín Hurtado Torrealba

GSkit: a domain-specific language and framework for Groth-Sahai proofs

More at benjamin.hurta.do/gskit

Is GSkit sound?

Three separate questions:

The proofs

GSkit generates standard Groth-Sahai proofs [GS08] over BN254 (SXDH): no new cryptography

With the soundness CRS the proofs are perfectly sound; with the WI CRS they are sound under SXDH

The language

Formal grammar and typing rules: a well-typed spec elaborates into a Groth-Sahai equation of the inferred type

That property is not formally proven: mechanizing it is future work

The implementation

The prover and the verifier come from the same elaboration, so they cannot disagree

BLS end to end: the valid signature verifies and a forged one is rejected, with both CRS types

GSkit proves exactly the equation you write. Whether that equation is the right one for the protocol is still up to whoever writes it.

WI or zero-knowledge?

In Groth-Sahai [GS08] every equation admits WI proofs, but only MSMEs, QEs and PPEs with $t_T = 1$ are ZK.

- The BLS proof we showed is a PPE with $t_T = e(\mathcal{H}(m), vk) \neq 1$: it is only WI.
- And BLS signatures are unique: with a single possible witness, WI does not hide it.

To make it ZK: also commit to $Y = \mathcal{H}(m)$ and prove two equations, a PPE with $t_T = 1$ and a linear MSME in $\mathbb{G}_1$, both ZK:

$$e(\sigma, g_2) \cdot e(Y, vk)^{-1} = 1_T \qquad Y = \mathcal{H}(m)$$

In the DSL that is one more witness and one MSME equation; the thesis typing rules cover both.

What does ABS-UCL assume against Sybil attacks?

The linking token is a weak Boneh–Boyen signature on the thread, made with the author's key:

$$\tau = \left[\frac{1}{usk + \text{thread}} \right] g$$

- It is deterministic: the same author in the same thread gets the same τ , so each person has **one voice per thread**.
- Tokens from different threads cannot be linked: the author stays anonymous.

Assumption: each person holds **a single key *usk***. The authority that issues the attributes must certify one key per person, e.g. one per national ID. Anyone who gets several keys gets several τ : ABS-UCL does not prevent that, because it is a registration problem, not a cryptographic one.

References

- [GS08]** J. Groth, A. Sahai. *Efficient Non-interactive Proof Systems for Bilinear Groups*. EUROCRYPT 2008, LNCS 4965, pp. 415–432.
- [BLS04]** D. Boneh, B. Lynn, H. Shacham. *Short Signatures from the Weil Pairing*. Journal of Cryptology 17(4), pp. 297–319, 2004.
- [BB04]** D. Boneh, X. Boyen. *Short Signatures Without Random Oracles*. EUROCRYPT 2004, pp. 56–73.
- [EKCGD14]** A. El Kaafarani, L. Chen, E. Ghadafi, J. Davenport. *Attribute-Based Signatures with User-Controlled Linkability*. CANS 2014, pp. 256–269.
- [EG17]** A. El Kaafarani, E. Ghadafi. *Attribute-Based Signatures with User-Controlled Linkability Without Random Oracles*. IMACC 2017, pp. 161–184.
- Figures** Ali Baba's cave illustrations by Dake, Wikimedia Commons, CC BY 2.5.