

CRIPTOGRAFÍA · BILINEAR MAPS · NIZK

GSkit

Un lenguaje de dominio específico y un *framework* para pruebas de Groth-Sahai

Benjamín Hurtado Torrealba

Profesores guía: Matías Toro Ipinza · Federico Olmedo Berón

Departamento de Ciencias de la Computación · Universidad de Chile

Magíster en Ciencias, mención Computación · Ingeniería Civil en Computación

MARTES 13 DE OCTUBRE DE 2026

benjamin.hurta.do/gskit

Agenda

1 Pruebas *zero-knowledge*

2 Firmas y *bilinear maps*

3 Pruebas de Groth-Sahai

4 GSkit: el DSL y el *framework*

5 Caso de estudio: ABS-UCL

6 Evaluación y conclusiones

¿Por qué probar sin revelar?

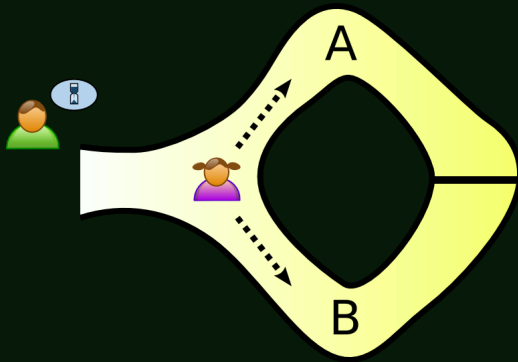
Todos los días tenemos que **convencer a alguien de un hecho**, y normalmente lo hacemos entregando la evidencia: el carnet para probar nuestra edad, una contraseña para probar que somos nosotros.

Pero muchas veces esa evidencia es un secreto que preferiríamos no entregar. ¿Podemos probar que el enunciado es verdadero **sin revelar el secreto**?

Esa es la pregunta que responden las pruebas *zero-knowledge*.

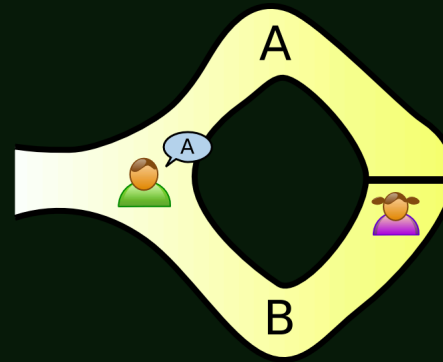
Zero-knowledge, con un ejemplo

Peggy (*prover*) afirma que conoce la palabra secreta que abre la puerta dentro de la cueva de Alí Babá. Victor (*verifier*) quiere convencerse sin aprender la palabra.



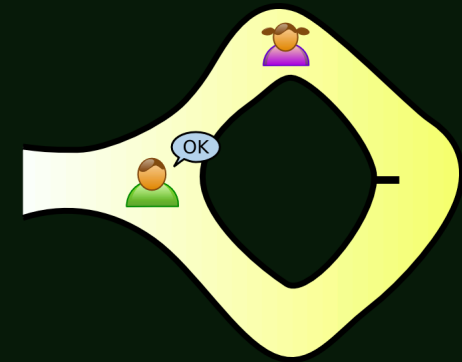
1 · Commit

Peggy entra por A o por B al azar, mientras
Victor espera afuera



2 · Challenge

Victor grita el lado por el que debe salir



3 · Response

Peggy sale por ese lado, cruzando la puerta
si es necesario

Tres propiedades

Repetir el juego de la cueva hace concretas las tres propiedades de una prueba *zero-knowledge*:

Completeness

Un *prover* honesto con un *witness* válido siempre convence al *verifier*

Si Peggy conoce la palabra, siempre sale por donde Victor pide

Soundness

Un *prover* tramposo no puede convencer al *verifier* de un enunciado falso

Sin la palabra la descubren la mitad de las veces: tras k rondas engaña a Victor con probabilidad 2^{-k}

Zero-knowledge

El *verifier* no aprende nada más allá de que el enunciado es verdadero

Victor solo la ve salir por donde pidió, una escena que podría haber montado él mismo

Formalmente — NIZK

Sea $\mathcal{R}$ una relación NP y el lenguaje que define:

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

donde x es el **enunciado** (público) y w el **witness** (secreto). Una prueba *non-interactive zero-knowledge* (NIZK) es un único mensaje π , que se verifica contra un *common reference string*:

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

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

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

● valores secretos: este color los marca en toda la presentación

El peor banco del mundo

Para hacerlo concreto, les presento al **peor banco del mundo**. Emite cheques, y todos a su alrededor necesitan una forma de distinguir uno auténtico de una falsificación.



Firmas, en una idea

Un **esquema de firma** es la herramienta del banco contra las falsificaciones. El banco guarda una clave secreta sk y publica una clave de verificación vk : cualquiera puede verificar, nadie puede falsificar.

- **KeyGen** — genera el par de claves (sk, vk) . El secreto se queda en el banco; vk se hace pública.
- **Sign** — solo quien tiene sk puede convertir un mensaje m en una firma σ .
- **Verify** — cualquiera con vk puede confirmar que σ corresponde a m .

El ingrediente que hace posible un **buen** esquema de firma es el *bilinear map*.

Bilinear maps

Un **bilinear map** es una función $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ que permite **mover escalares a través de un pairing**:

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

- $\mathbb{G}_1, \mathbb{G}_2$ y $\mathbb{G}_T$ son grupos finitos de orden primo p , definidos sobre curvas elípticas
- g_1, g_2 generan $\mathbb{G}_1$ y $\mathbb{G}_2$ respectivamente, y $a, b \in \mathbb{Z}_p$

Esta única propiedad, sacar los exponentes de un *pairing*, es la palanca detrás de las firmas BLS y de las pruebas de Groth-Sahai.

Firma BLS

Boneh–Lynn–Shacham [BLS04] convierten esa palanca en un esquema de firma: la clave secreta se esconde como **exponente**, y el *pairing* permite que cualquiera la verifique **sin verla nunca**. Una función *hash-to-curve* $\mathcal{H} : \{0, 1\}^* \rightarrow \mathbb{G}_1$ lleva los mensajes a la curva.

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

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

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

La verificación funciona porque la bilinealidad hace que ambos lados colapsen al mismo valor:

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

El peor banco del mundo usa BLS

Así que el banco adopta BLS: emite un cheque y lo **firma** con su clave secreta. Cualquiera puede verificarlo; solo el banco pudo haberlo producido.



Vale por 1_000_000

σ



Tu peor amigo

mirando...

El banco **firma** el cheque con BLS:

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

Cualquiera con la clave de verificación vk puede verificarlo:

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

Con solo **ver** σ , tu peor amigo puede **cobrar el cheque**: el banco le paga a quien presente una firma válida.

Un nuevo problema

El cheque está firmado y es verificable. Ahora **tienes un cheque válido** y quieres probarle a un amigo que *el banco realmente lo firmó*, sin entregarle σ , porque si no podría cobrarlo él mismo.

Quieres probar “*existe un σ válido*”, es decir, que la ecuación de verificación **tiene solución**, manteniendo σ oculto.

Eso es exactamente una **prueba zero-knowledge sobre una ecuación con bilinear maps**. Aquí entra Groth-Sahai.

Pruebas de Groth-Sahai

Las pruebas de Groth-Sahai [GS08] son **pruebas *non-interactive zero-knowledge*** eficientes para enunciados escritos como **ecuaciones sobre un *bilinear group***. El tipo más expresivo es la *pairing-product equation* (PPE):

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

CONSTANTES · PÚBLICAS

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

WITNESS · SECRETO

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

El *prover* conoce un *witness* que satisface la ecuación; el *verifier* se convence **sin llegar a conocerlo**.

Cuatro tipos de ecuaciones

La misma forma cuadrática, sobre distintos *bilinear maps*, da **cuatro tipos de ecuaciones**:

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 EN $\mathbb{G}_1$

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

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

MSME · MULTI-SCALAR EN $\mathbb{G}_2$

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

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

QE · QUADRATIC EN $\mathbb{Z}_p$

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

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

Formalmente — el lenguaje de Groth-Sahai

$$\mathcal{L} := \left\{ x \mid \exists w : \mathcal{E}_1(x, w), \dots, \mathcal{E}_n(x, w) \text{ se cumplen} \right\}$$

donde cada $\mathcal{E}_i$ es uno de los **cuatro tipos de ecuaciones de Groth-Sahai** sobre las variables:

Variables de grupo

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

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

Variables escalares

$$x_1, \dots, x_{m'} \in \mathbb{Z}_p$$

$$y_1, \dots, y_{n'} \in \mathbb{Z}_p$$

Una sola prueba muestra que *todas* las ecuaciones se cumplen para un mismo *witness*. El DSL de GSKit, que viene a continuación, es una sintaxis para exactamente este lenguaje: declaras el *witness* y las ecuaciones, y el *framework* construye la prueba.

El problema que queremos resolver

Escribir una de estas pruebas **a mano** es intrincado y propenso a errores: hay que hacer un *commitment* de cada variable, construir los vectores a , b y la matriz γ , y mantener perfectamente sincronizadas las vistas del *prover* y del *verifier*.

La dificultad

Commitments de bajo nivel, fáciles de equivocar y difíciles de leer; *prover* y *verifier* pueden desincronizarse.

La idea

Describir *qué* es la ecuación, de forma declarativa, y dejar que un compilador genere el resto.

Por eso diseñamos un pequeño **DSL**: declaras la ecuación una vez, y el *framework* compila tanto la prueba como la verificación.

Trabajo relacionado — y el vacío

DSLs *zero-knowledge*

Circom · ZoKrates · Noir · Cairo · Leo

Compilan programas a **circuitos** (R1CS, ACIR, AIR) para *back-ends* SNARK y STARK. Ninguno soporta de forma nativa ecuaciones sobre *bilinear groups*.

Bibliotecas de Groth-Sahai

groth-sahai-rs · groth-sahai-python · un prototipo en Java

APIs programáticas: el desarrollador todavía clasifica los *witnesses*, elige los tipos de ecuación, construye los *commitments* y nombra las variables entre subpruebas.

GSkit llena ese vacío: enunciados de Groth-Sahai escritos como **ecuaciones** declarativas, no como circuitos ni llamadas a una API.

Esta tesis

- 1 Un DSL para enunciados de Groth-Sahai.** Se declaran variables, constantes y ecuaciones; un sistema de tipos infiere el tipo de cada ecuación (PPE, MSME, QE) y rechaza especificaciones mal formadas *antes* de construir cualquier prueba.
- 2 GSKit, un *framework* en Python** sobre Charm/PBC que compila una especificación en **ambos**, el *prover* y el *verifier*, desde la línea de comandos o desde código Python con `@gsfy`.
- 3 Dos casos de estudio**, BLS y ABS-UCL; el segundo, comparado con una implementación anterior escrita a mano.

Un DSL para Groth-Sahai

Una especificación tiene solo **tres bloques**: qué nombres están **ocultos** (variables), cuáles son **públicos** (constantes) y las **ecuaciones**:

variables:

```
<nombres ocultos> : G1 | G2 | ZP
```

constants:

```
<nombres públicos> : G1 | G2 | ZP | GT
```

equations:

```
<término> + <término> + ... = <nombre>
```

variables — el *witness*, lo que se mantiene secreto (solo `G1`, `G2`, `ZP`)

constants — valores públicos, incluido el lado derecho (aquí se permite `GT`)

equations — sumas de términos producto; `*` es la operación bilineal y `+` combina términos

Cómo se lee un término

Cada término es un producto de nombres. **En qué grupos viven los nombres** decide el tipo de ecuación, y el compilador lo infiere por ti:

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

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

Prueba BLS para mi peor amigo

Volvamos al cheque. **Tú** (el *prover*) conoces σ ; tu amigo (el *verifier*) debe convencerse sin verla.



Idea clave: tu amigo aprende que *existe una firma válida*, pero nunca ve σ .

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

La prueba BLS, en el DSL

Todo el enunciado “*conozco un σ tal que $e(\sigma, g_2) = t_T$* ” se reduce a unas pocas líneas. El secreto σ es una variable; todo lo demás es una constante:

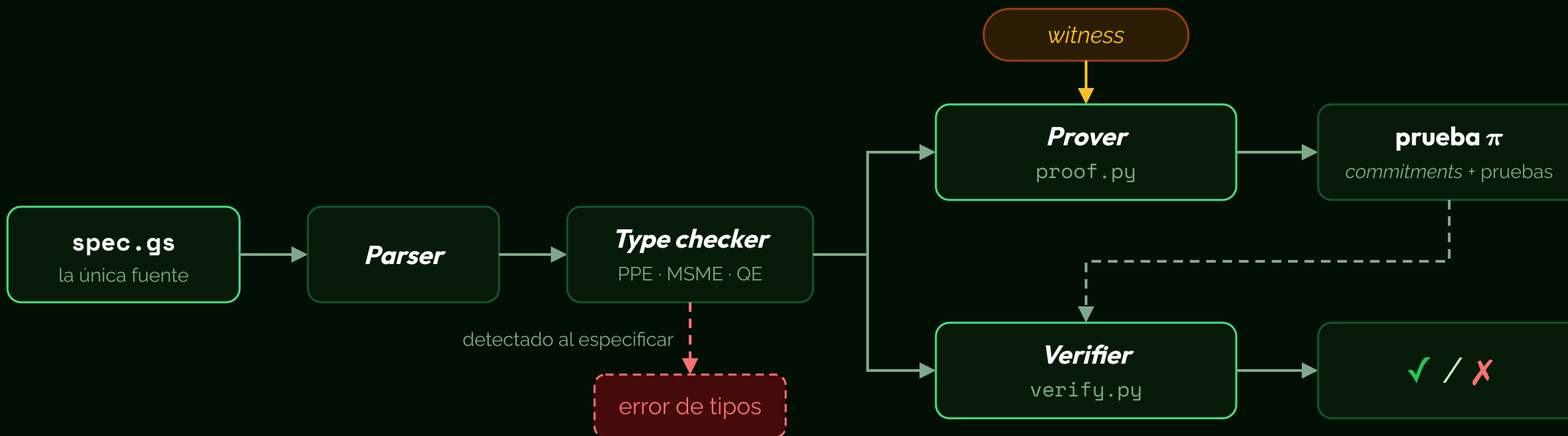
- σ es el *witness* → se declara en `variables`
- g_2 y el objetivo t_T son *públicos* → `constants`
- una sola **PPE**: `signature * g2 = rhs`

El compilador lee los tipos y reconoce una *pairing-product equation*, sin configuración manual.

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

Una especificación, ambos lados

La misma especificación tipada se compila en **dos** programas, así que las vistas de las ecuaciones del *prover* y del *verifier* nunca pueden desincronizarse:



Dentro de Python, el decorador `@gsfy` lee la misma especificación desde un `GS_STRING` en la función y conecta sus variables automáticamente.

Donde todo empezó: un foro anónimo

BLS fue el calentamiento. Esta tesis partió de un objetivo concreto: un **foro anónimo**, donde los miembros prueban que pertenecen sin revelar quiénes son. Pero el anonimato por sí solo tiene un problema:

FORO ANÓNIMO · #PARQUE 4 MENSAJES

	anon_4821	En mi opinión, deberíamos patear cachorros en el parque.
	anon_1337	Totalmente de acuerdo. Por fin alguien lo dice.
	anon_0042	+1. Esos cachorros se lo buscaron.
	amo_los_perros	Como amante de los perros, apoyo esto totalmente.

▲ 4 a favor · 0 en contra



No es solo un problema de foros: las cuentas títere y las granjas de bots fabrican consenso en el debate público y en las elecciones de hoy, una amenaza real para la democracia.

La solución: ABS-UCL

Attribute-Based Signatures with User-Controlled Linkability [EKCGD14, EG17]. Cada mensaje lleva una firma que prueba:

- “*mis atributos cumplen la política*” (por ejemplo, *estudiante de la UChile*), sin revelar quién firmó ni qué atributos usó;
- un **linking token** τ bien formado, derivado de la clave secreta del autor *usk* y del hilo.

#PARQUE

FIRMADO CON ABS-UCL

τ 9c4e...a1	En mi opinión, deberíamos patear cachorros en el parque.
τ 9c4e...a1	Totalmente de acuerdo. Por fin alguien lo dice.
τ 9c4e...a1	+1. Esos cachorros se lo buscaron.
τ 9c4e...a1	Como amante de los perros, apoyo esto totalmente.

4 mensajes · 1 autor

Mismo autor, mismo hilo, mismo τ : los cuatro “partidarios” cuentan como **una sola voz**. Entre hilos distintos los *tokens* no se pueden relacionar, así que los autores siguen siendo anónimos. Privacidad **y** rendición de cuentas, en una sola firma.

Una firma, cuatro subprotocolos

Una sola firma ABS-UCL es una **conjunción** de enunciados *zero-knowledge*, cada uno de un subprotocolo independiente: muchas ecuaciones de Groth-Sahai probadas como una sola.

RPSPS

Las claves de atributos del usuario fueron **realmente emitidas** por las autoridades.

Re-randomizable structure-preserving signatures

MSP

Esos atributos **cumplen la política de acceso**.

Monotone span program

LIT

El **linking token** está bien formado: la clave de la *linkability* controlada.

Linking token

FBB · PSDO

Un **pseudoatributo** aporta una rama distinguida, así que la política siempre se puede satisfacer.

Firma Full Boneh–Boyen [BB04]

La recompensa: componer, no duplicar

Cada subprotocolo declara su lógica de Groth-Sahai **una sola vez** con `@gsfy`; el esquema compone esas especificaciones sobre un contexto compartido, y tanto el *prover* como el *verifier* salen de esa única fuente.

Antes · a mano



cada ecuación se escribe dos veces, y las copias pueden divergir

GSkit · modular



cada ecuación se escribe una vez, dentro de su subprotocolo

El resultado

La evidencia decisiva no es tener menos líneas en total, sino la **eliminación del código de ecuaciones duplicado y mantenido a mano**:

Esfuerzo para construir las ecuaciones	A mano	GSkit (modular)
Líneas en <code>sign()</code>	85	8 ecuaciones declarativas
Líneas en <code>verify()</code>	84	0, generado
Objetos de ecuación y de <i>pairing</i> contruidos a mano	60	0
Veces que se escribe cada ecuación	2	1

El código total escrito por el desarrollador no es menor (420 → 540 líneas): cada subprotocolo ahora trae su propia especificación.

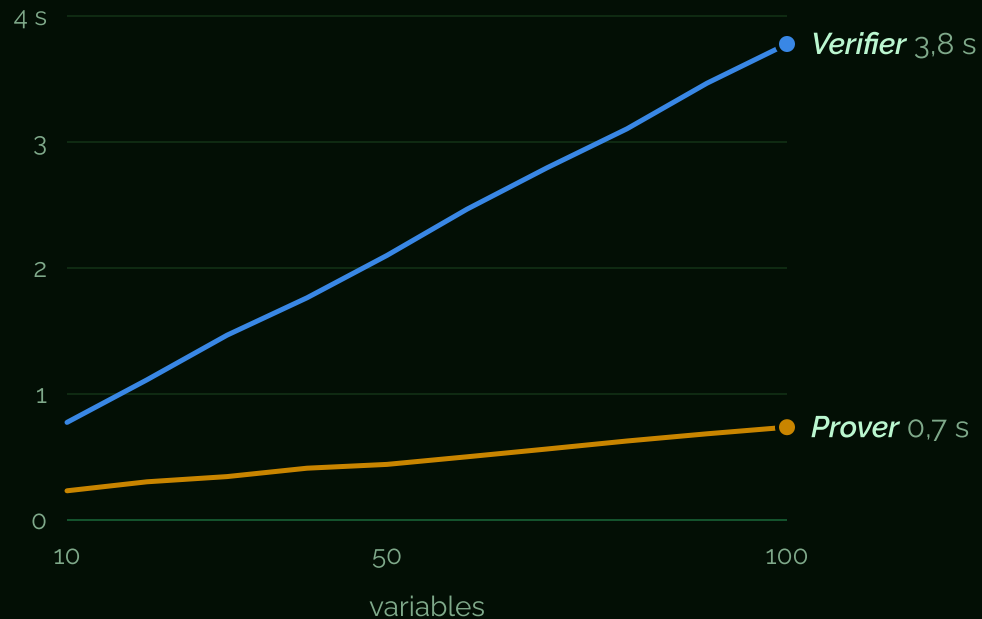
La contribución: llevar las ecuaciones a una única fuente declarativa es lo que reduce la superficie de error, y lo que hace práctico construir y mantener un esquema tan complejo como ABS-UCL.

Rendimiento

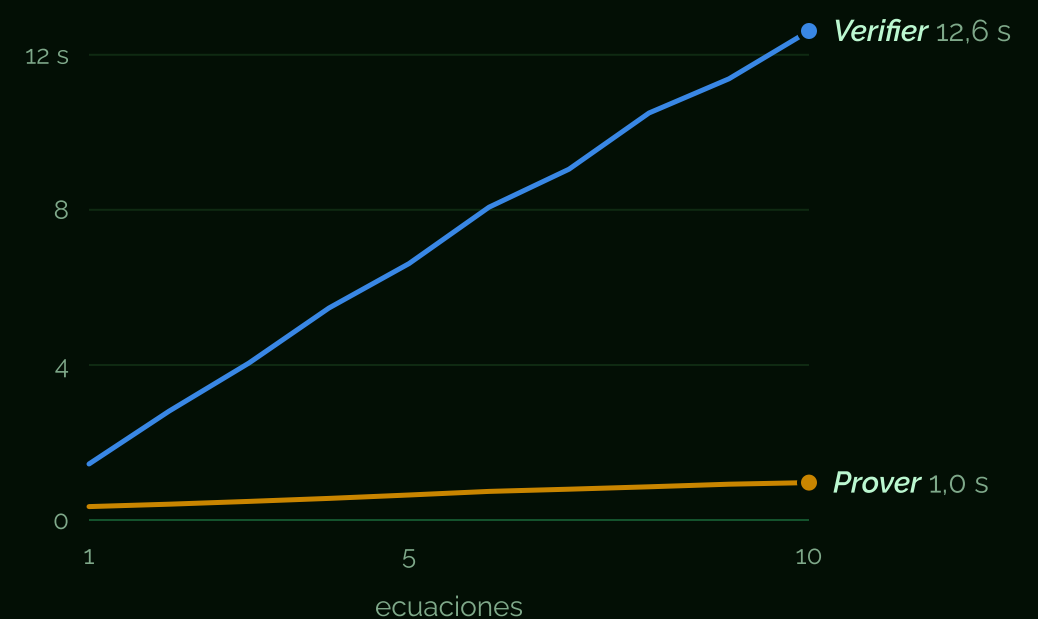
Generar y verificar una prueba de *pairing-product equations* toma tiempo **lineal** en el número de variables y en el número de ecuaciones.

— Verifier — Prover

Aumentando el número de variables



Aumentando el número de ecuaciones



GSkit sobre Charm/PBC, curva BN254. Datos de la evaluación de la tesis.

Conclusiones

Lo que mostramos

- Una especificación declarativa por enunciado: el *prover* y el *verifier* se generan a partir de ella, así que no pueden desincronizarse.
- El chequeo de tipos mueve una clase de errores del tiempo de ejecución al **tiempo de especificación**.
- En ABS-UCL, 16g líneas escritas a mano y 60 objetos de ecuación se convierten en **8 ecuaciones declarativas**, escritas una sola vez.

Trabajo futuro

- Verificación en lote y optimizaciones del tamaño de la prueba
- Más casos de estudio, además de BLS y ABS-UCL
- De punta a punta: el foro anónimo basado en atributos que motivó este trabajo

Hipótesis confirmada: un lenguaje de dominio específico baja la barrera para construir pruebas de Groth-Sahai, y reduce el esfuerzo y la superficie de error sin renunciar a sus garantías de seguridad.

Gracias

¿Preguntas?



Benjamín Hurtado Torrealba

GSkit: un lenguaje de dominio específico y un *framework* para pruebas de Groth-Sahai

Más información en benjamin.hurta.do/gskit

¿Es *sound* GSkit?

Conviene separar tres preguntas:

Las pruebas

GSkit genera pruebas estándar de Groth-Sahai [GS08] sobre BN254 (SXDH): no agrega criptografía nueva

Con el CRS de *soundness*, las pruebas son perfectamente *sound*; con el CRS WI, son *sound* bajo SXDH

El lenguaje

Gramática y reglas de tipado formales: una especificación bien tipada se elabora en una ecuación de Groth-Sahai del tipo inferido

Esa propiedad no está demostrada formalmente: mecanizarla es trabajo futuro

La implementación

El *prover* y el *verifier* salen de la misma elaboración, así que no pueden discrepar entre sí

BLS de punta a punta: la firma válida verifica y una falsificada se rechaza, con ambos tipos de CRS

GSkit prueba exactamente la ecuación que escribes. Que esa ecuación sea la correcta para el protocolo sigue siendo responsabilidad de quien la escribe.

¿WI o zero-knowledge?

En Groth-Sahai [GS08] todas las ecuaciones admiten pruebas WI, pero solo son ZK las MSME, las QE y las PPE con $t_T = 1$.

- La prueba BLS que mostramos es una PPE con $t_T = e(\mathcal{H}(m), vk) \neq 1$: solo es WI.
- Además, la firma BLS es única: con un solo *witness* posible, WI no garantiza ocultarlo.

Para que sea ZK: se compromete también $Y = \mathcal{H}(m)$ y se prueban dos ecuaciones, una PPE con $t_T = 1$ y una MSME lineal en $\mathbb{G}_1$, ambas ZK:

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

En el DSL son un *witness* más y una ecuación MSME; las reglas de tipado de la tesis las cubren.

¿Qué supone ABS-UCL contra el ataque Sybil?

El *linking token* es una firma Boneh–Boyen débil del hilo, hecha con la clave del autor:

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

- Es determinista: el mismo autor en el mismo hilo obtiene el mismo τ , así que cada persona tiene **una voz por hilo**.
- Entre hilos distintos los τ no se pueden relacionar: el autor sigue siendo anónimo.

Supuesto: cada persona tiene **una sola clave** *usk*. La autoridad que emite los atributos debe certificar una clave por persona, por ejemplo una por RUT. Si alguien consigue varias claves, obtiene varios τ : ABS-UCL no lo impide, porque es un problema de registro y no criptográfico.

Referencias

- [GSo8]** 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.
- Figuras** Ilustraciones de la cueva de Alí Babá: Dake, Wikimedia Commons, CC BY 2.5.