

Algorand Community Study Group

1: Elliptic Curve Cryptography

HashMapsData2Value

Digital Community Champion
Algorand Foundation

May 2, 2023

Welcome!

- Welcome!
- Study group? Discord: #cryptography-study-group
- GitHub Repo:
<https://github.com/HashMapsData2Value/Algorand-Community-Study-Group>
- Please introduce yourselves
- Topic - Elliptic Curve Cryptography
- Chapter 15 in *A Graduate Course in Applied Cryptography* by Dan Boneh and Victor Shoup (version 0.6)

Agenda

1. Chapter 15

15.1 The group of points of an elliptic curve

15.2 Elliptic curves over finite fields

15.3 Elliptic curve cryptography

15.4 Pairing based cryptography

Intermission: AVM EC Math Pull Request

15.5 Signature schemes from pairings

15.6 Advanced encryption schemes from pairings

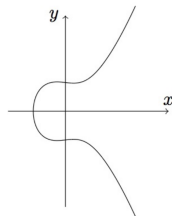
15.7 The functional encryption paradigm

15.8 Multilinear Maps

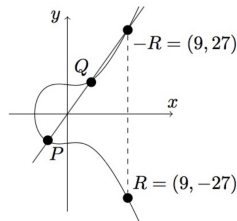
2. Next topic?

15.1 The group of points of an elliptic curve

- Diophantus, $y^2 = x^3 - x + 9$ (Rational)
- Points: (x,y)
- Chord Method, $U \boxplus V = -W$, $U \neq -V$
- Tangent Method, $P \boxplus P = Q$, $y \neq 0$



(a) The curve



(b) Adding $P = (-1, -3)$ and $Q = (1, 3)$

Figure: Credits [Boneh, Shoup, 2023].

15.2 Elliptic curves over finite fields

- Rationals $\mathbb{Q} (1, 4.2, \frac{9}{36}, -3) \rightarrow$ Finite Field $\mathbb{F} (... , -1, 0, 1, 2, ... \text{ mod } p) \mathbb{Z}/p\mathbb{Z}$
- E/\mathbb{F}_p - denotes elliptic curve defined over finite field
- $E(\mathbb{F}_{p^e})$ - denotes the set of all points on curve + the "point at infinity \mathcal{O} "
- Example: $E: y^2 = x^3 + 1, p = 11 \rightarrow$
 $E(\mathbb{F}_{11}) = \{\mathcal{O}, (-1, 0), (0, \pm 1), (9, \pm 2), (6, \pm 3), (8, \pm 4), (3, \pm 5)\}$
- $|E(\mathbb{F}_{11})| = 12$ "group elements"
- Group Operation - \boxplus . \mathcal{O} is identity element $P \boxplus \mathcal{O} = P$
- Pick one point to be a "generator"! Will generate a group of certain order.
- Two notations: $\underbrace{G + G + \dots + G}_{\text{a additions}} = A, \underbrace{g * g * \dots * g}_{\text{a multiplications}} = g^a.$
- Forms: Weierstrass, Edwards, Montgomery

15.3 Elliptic curve cryptography

- Let $G, g \in E(\mathbb{F}_{p^e})$ of order q (i.e. generates group of q number points on curve). If q is itself prime, then:
 - $qG \equiv \mathbb{O}$ (additive) or $g^q \equiv \mathbb{O}$ (multiplicative)
 - Discrete Log Problem: $\alpha G \equiv A \rightarrow \alpha \equiv A/G$ or $g^\alpha \equiv a \rightarrow \alpha \equiv \log(a)_g$ HARD!
- Need to be careful! You're in trouble if:
 - $|E(\mathbb{F}_p)| = p$, i.e. there are as many unique points on the curve as we have integers in our finite field, or
 - There exists a small integer τ such that $|E(\mathbb{F}_p)|$ divides $p^\tau - 1$.
- So we just stick to standard curves and parameters, e.g:
 - P256 - Shady! NSA didn't explain a certain parameter...
 - Curve25519 - Algorand!
 - Ed25519, refer to explanation in the repo notes under #ed25519-signature.
 - With signatures: scalar α is secret key, $\alpha G = A$ is public key. Secret key signs transactions from public key (address).

15.4 Pairing based cryptography - Definition

Theorem (Definition 15.2. in [Boneh, Shoup, 2023])

Let $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T$ be three cyclic groups of prime order q where $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ are generators. A pairing is an efficiently computable function $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ satisfying the following properties:

1. *bilinear*: for all $u, u' \in \mathbb{G}_0$ and $v, v' \in \mathbb{G}_1$ we have
 - 1.1 $e(u \cdot u', v) = e(u, v) \cdot e(u', v)$, and
 - 1.2 $e(u, v \cdot v') = e(u, v) \cdot e(u, v')$
 2. *non-degenerate*: $g_T := e(g_0, g_1)$ is a generator of \mathbb{G}_T (and doesn't just output $1 \in \mathbb{G}_T$ for all inputs)
- If $\mathbb{G}_0 = \mathbb{G}_1$ pairing is "symmetric", otherwise asymmetric. \mathbb{G}_0 and \mathbb{G}_1 are called the "pairing groups" and \mathbb{G}_T the target group.
 - (Obviously) we construct pairings from elliptic curves.

15.4 Pairing based cryptography - Groups

- \mathbb{G}_0 is order q subgroup of $E(\mathbb{F}_p)$, for some prime q .
- \mathbb{G}_1 is order q subgroup of $E(\mathbb{F}_{p^d})$, for some $d > 0$ (embedding degree of curve) where $\mathbb{G}_1 \cap \mathbb{G}_0 = \{\mathcal{O}\}$
- \mathbb{G}_T is an order q multiplicative subgroup of the finite field \mathbb{F}_{p^d} .
- Note that as \mathbb{G}_0 is defined over base field \mathbb{F}_p and \mathbb{G}_1 over a larger field \mathbb{F}_{p^d} points in \mathbb{G}_0 are "smaller" We want d to be not too large as we'd get a huge \mathbb{G}_1 . $d < 16$ is "pairing friendly".
- Central property: $e(g_0^\alpha, g_1^\beta) = e(g_0, g_1)^{\alpha \cdot \beta} = e(g_0^\beta, g_1^\alpha)$
- Pairing function e - Weil pairing (variant: Tate and Ate pairings), efficiently evaluated with Miller's algorithm.

15.4 Pairing based cryptography - DDH

- Pairings were originally thought to be BAD! Why?
- When $\mathbb{G}_0 = \mathbb{G}_1$ decision Diffie-Hellman (DDH) problem in \mathbb{G}_0 no longer HARD
- DDH: For $\alpha, \beta, \delta \xleftarrow{\$} \mathbb{F}_p$ (secret) and $g_0^\gamma = g_0^{\alpha * \beta}$, g_0^δ and g_0^γ both look "random". I.e. shouldn't be possible for an "outsider" to decide whether δ or $\gamma = \alpha * \beta$...
- However, $e(g_0^\alpha, g_0^\beta) \stackrel{?}{=} e(g_0, g_0^\gamma) \rightarrow e(g_0, g_0)^{\alpha * \beta} = e(g_0, g_0)^\gamma$
- (Note: Not to be confused with $\gamma = \alpha + \beta$ as checking $g^\gamma = g^\alpha * g^\beta = g^{\alpha + \beta}$ is trivial.
- This makes e.g. ElGamal encryption scheme insecure on \mathbb{G}_0 .
- For $\mathbb{G}_0 \neq \mathbb{G}_1$ asymmetric pairings DDH assumption might hold in both \mathbb{G}_0 and \mathbb{G}_1 . It does for "the most efficient asymmetric pairings".
- Also, we need to pick groups that ensure discrete log problem is HARD in \mathbb{G}_T , otherwise it breaks down in \mathbb{G}_0 and \mathbb{G}_1 . ($u := e(g_0^\alpha, g_1) = e(g_0, g_1)^\alpha = g_T^\alpha$. If $\alpha = \log(u)_{g_T}$ easy...)

15.4 Pairing based cryptography - Curves

name	d	q group order	\mathbb{F}_p size	\mathbb{G}_0 bytes	\mathbb{G}_1 bytes	advantage
bn256	12	256	256	32	64	Speed
bls381	12	256	381	48	96	Security

Table: Comparison between curves bn256 and bls381. They are also called BN254 and BLS 12-381 respectively, e.g. in AVM docs

- bn256 faster due to smaller finite field it is defined over, while bls381 more secure as the DLOG problem in \mathbb{G}_T is harder.

15.4 Pairing based cryptography - Speed Tricks

- Exponentiation in \mathbb{G}_0 fastest (0.22 ms), then \mathbb{G}_1 (0.44 ms), then \mathbb{G}_T (0.95 ms). Pairing $\mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ (2.32 ms).
- $e(g_0^\alpha, g_1) = e(g_0, g_1^\alpha) = e(g_0, g_1)^\alpha$ - same results, different computation speeds.
- A product of pairings $\prod_{i=1}^n e(u_i, v_i) \in \mathbb{G}_T$ can also be sped up by aggregating final exponentiation. E.g $n = 3$:
 - $u_1, u_2, u_3 \in \mathbb{G}_0 \rightarrow g_0^\alpha, g_0^\beta, g_0^\gamma$
 - $v_1, v_2, v_3 \in \mathbb{G}_1 \rightarrow g_1^\delta, g_1^\epsilon, g_1^\zeta$
 - $e(g_0^\alpha, g_1^\delta) e(g_0^\beta, g_1^\epsilon) e(g_0^\gamma, g_1^\zeta) = e(g_0, g_1)^{\alpha\delta} e(g_0, g_1)^{\beta\epsilon} e(g_0, g_1)^{\gamma\zeta} = e(g_0, g_1)^{\alpha\delta + \beta\epsilon + \gamma\zeta}$
- (Provides intuition for how Groth16 ZK-SNARKs are so efficient.)

AVM EC Math Pull Request

- <https://github.com/algorand/go-algorand/pull/4924> (Liable to change!)
- New opcodes (each with option for BN254 or BLS318, \mathbb{G}_0 or \mathbb{G}_1):
 - ec_add: $A + B = C, \in \mathbb{G}$ (equivalent to $g^a * g^b = g^c$)
 - ec_scalar_mul: $\alpha P = A \in \mathbb{G}$ (equivalent to $(g)^a$)
 - ec_multi_exp: multiple inputs version of ec_scalar_mul.
 - ec_pairing_check: Check if $e(A_1, B_1)e(A_2, B_2)...e(A_n, B_n) = 1$.
 - ec_subgroup_check: check if point is "valid" ($\in \mathbb{G}$)
 - ec_map_to: maps input to point (i.e., $H(m) \in \mathbb{G}$)
- Pairing check VERY expensive, only possible in logic sig (stateless smart contract)

15.5 Signature schemes from pairings - BLS

Theorem (Boneh-Lynn-Shacham (BLS) Signature Scheme)

BLS algorithms:

- *Key Generation:* $sk := \alpha \xleftarrow{\$} \mathbb{Z}_q, pk := g_1^\alpha \in \mathbb{G}_1$.
 - *Sig. Generation:* $\sigma \leftarrow H(m)^\alpha \in \mathbb{G}_0$
 - *Sig. Verification:* $e(H(m), pk) \stackrel{?}{=} e(\sigma, g_1)$
-
- Hash function $H : \mathbf{M} \rightarrow \mathbb{G}_0$ (maps input directly to \mathbb{G}_0 , a point.)
 - Signature σ a point in \mathbb{G}_0 , pk a point in \mathbb{G}_1 .
 - $e(H(m), pk) = e(H(m), g_1^\alpha) = e(H(m)^\alpha, g_1) = e(\sigma, g_1)$

15.5 Signature Schemes from pairings - BLS Advantages

- Since $\sigma \in \mathbb{G}_0$, the signature is short (32 bytes for bn256)!
 - Compare with 64 bytes signatures in Ed25519, though without expensive pairing operation.
 - BLS possible to swap s.t. $pk \in \mathbb{G}_0$ and $\sigma \in \mathbb{G}_1$ - short public key instead.
- BLS applications:
 - Signature Aggregation
 - Verify many signatures with few operations
 - Threshold Signatures
 - t out of n signers are enough to sign message
 - Blind Signatures
 - Signer signs encrypted message, but message integrity verifiable.
 - Distributive generation of secret key across several parties

15.5.2 Signature Aggregation

Theorem (BLS Signature Aggregation)

Given $\mathbf{pk} = (pk_1, \dots, pk_n) \in \mathbb{G}_1^n$, messages $\mathbf{m} = (m_1, \dots, m_n)$ and signatures $\sigma = (\sigma_1, \dots, \sigma_n) \in \mathbb{G}_0^n$, we have the following algorithms:

- *Gen. Sig. Aggregate:* $\sigma_1 * \sigma_2 * \dots * \sigma_n \rightarrow \sigma_{ag}$ (short)
- *Ver. Sig. Aggregate:* $e(\sigma_{ag}, g_1) = e(H(m_1), pk_1) * \dots * e(H(m_n), pk_n)$
- $e(\sigma_{ag}, g_1) = e(\sigma_1 * \dots * \sigma_n, g_1) = \prod_{i=1}^n e(\sigma_i, g_1) = \prod_{i=1}^n e(H(m_i)_i^\alpha, g_1) = \prod_{i=1}^n e(H(m_i), g_1^\alpha i) = \prod_{i=1}^n e(H(m_i), pk_i)$
- Remember: aggregation doesn't have to be "one-time" process, it can happen gradually.
- If $m_1 = m_2 = \dots = m_n$ (single message) $e(\sigma_{ag}, g_1) = e(H(m), apk)$, $apk := pk_1 * \dots * pk_n$.
 apk is an aggregate public key, can be pre-computed.

15.5.2 Signature Aggregation - Rogue Key

WARNING: Rogue Key Attack!

BLS signature aggregation scheme works but is susceptible to rogue key attack, where an adversary targeting Bob can maliciously create a key that "negates" other keys in the aggregated signature.

The adversary can make it seem like Bob signed something he actually did not.

Thus, if you decide to implement BLS signature aggregation you must use one of two methods to make it secure:

- Message Augmentation
- Proof of Possession of Secret Key

15.6 Advanced encryption schemes from pairings - IBE

Identity-based Encryption (IBE)

"An Identity Base Encryption (IBE) scheme is a public-key cryptosystem where any string is a valid public key. In particular, email addresses and dates can be public keys." - Stanford

- Requires "Private Key Generator" (PKG), trusted coordinator with master secret key. With msk you can decrypt everything, but possible to distribute it across multiple parties.
- Advantages?
 - Saves DB space! (We already store usernames!)
 - Expiration: $pk := \text{user@email.com} || [\text{today}]$. (New secret key every day!)
 - Delegation: e.g. use email subject line as pk , different labels \rightarrow different decryption keys, given to different people.
- Paper: BasicIdent - [Boneh, Franklin].

15.7 The functional encryption paradigm - Levels

- Basic level: 1 key pair for encryption/decryption. All or nothing.
- Next level: (IBE) 1 master key pair + encrypt/decrypt by ID key pair
- Can we GENERALIZE this and obtain a broad definition of public-key encryption?
 - $(pk, msk) \leftarrow Setup(1^\lambda)$ - creates public key and master secret key.
 - $sk_f \leftarrow KeyGen(msk, f)$ - generates functional sk_f for a function f using msk .
 - $c \leftarrow Enc(pk, m)$ - uses pk to encrypt input x .
 - $y \leftarrow Dec(sk_f, c)$ - uses sk_f to calculate $y = f(x)$.
- In fact...
 - For "basic", f is identity function I . $msk = sk_f$ (only one functional sk), and it decrypts any well-formed ciphertext.
 - For IBE, there is a $sk_{id'}$ for every $id' \in ID$. c can be decrypted by $sk_{id'}$ if c was encrypted against id' .

15.7 The functional encryption paradigm - Examples

- Linear functional encryption
 - Doesn't strictly require pairings...
 - Allows for linear fitting on encrypted data, e.g. medical data.
 - Can give access to specific subset of points.
- Quadratic functional encryption
 - Allows for calculating variance and standard deviation on encrypted data.
- Attribute-based encryption
 - Instead of polynomials, predicate functions: $f : x \rightarrow \{0, 1\}$ (and families of a set of them).
 - Ciphertext $c = (x, m)$ - an encryption of attribute x and message m .
 - Decryption is dependent on the secret key matching the attribute of the data (predicate function gives 1).

15.8 Multilinear Maps

- Bilinearity is *really* nice. Wouldn't it be nice if we could get multilinearity?
 - $e(g_0^{\alpha_0}, g_1^{\alpha_1}, \dots, g_n^{\alpha_n}) = e(g_0, g_1, \dots, g_n)^{\prod_{i=0}^n \alpha_i}$
 - ($g_T := e(g, \dots, g)$ a generator of \mathbb{G}_T .)
- Active field of research...
- Example application: Secret Sharing (you're the α)
 - 2 parties: $g^\alpha, g^\beta \rightarrow (g^\beta)^\alpha = g^{\alpha\beta}$
 - 3 parties: $g^\alpha, g^\beta, g^\gamma \rightarrow e(g^\beta, g^\gamma)^\alpha = e(g, g)^{\alpha\beta\gamma} = g_T^{\alpha\beta\gamma}$
 - 4 parties: [No *practical* solution....]



Anything else?

Did we forget something?

Next topic?

- The Smörgåsbord of inspiration:
 - Lattice-based Cryptography (State Proofs/FALCON Keys, Learning with Errors, FHE?)
 - Monero's Tech (Ring Signatures, Bulletproofs, ...)
 - ZK-SNARKs (Groth16, PLONK, ...)
 - Casino games with hidden state (IOHK-sponsored Royale, Kaleidoscope, 21; ZK proof of correctness of shuffles...)
 - Algorand Foundation's research into SMPC (SPRINT, TCaaS, NI-PVSS)
- Anything else?
- We can decide details on Discord...

References

-  Dan Boneh and Victor Shoup (2023 v0.6)
A Graduate Course in Applied Cryptography
cryptobook.us 15, 615 – 681.
-  Dan Boneh and Matthew Franklin (2003)
Identity based encryption from the Weil pairing
SIAM J. of Computing Vol. 32, No. 3, pp. 586-615, 2003

The End (for now!)