21^{st} Century Cryptography

- Everything we have talked about in the course thus far is 20^{th} century crypto (e.g. symmetric cryptography, DDH, RSA)
- Today: 21^{st} century cryptography - very powerful and surprising primitives

motivate via 2-party DH → (tripartite DH?) (Shamir's IBE problem)

Pairing-based cryptography: new algebraic structure on elliptic curve groups

Abstractly: G, G_T be finite cyclic groups of prime order q

Def. A pairing $e: G \times G \to G_T$ is an efficiently-computable mapping with the following properties:

- Efficient: the pairing e can be computed in polynomial time (non-trivial property)
- Bilinear: $\forall a, b \in \mathbb{Z}_q$ and $g \in G$: $e(g^a, g^b) = e(g,g)^{ab}$
- Non-degenerate: if g generates G, then $e(g,g)$ generates G_T
  - ↳ otherwise, consider mapping $e(g,g) \to 1_T$ (identity in G_T)

20^{th} century crypto: linear function in exponent
21^{st} century crypto: quadratic functions in exponent

Certain elliptic curve groups have efficiently computable pairings (Weil pairing / Tate pairing)

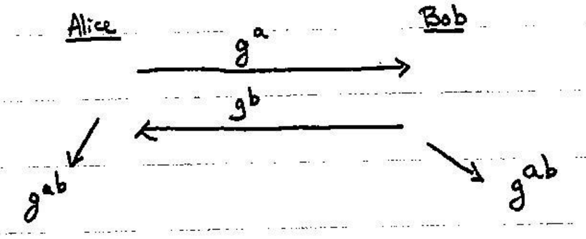
- Suppose $e: G \times G \to G_T$ has a pairing $\Rightarrow$ DDH in G is false
  - given (g, g^a, g^b, g^c), test if $e(g, g^c) \stackrel{?}{=} e(g^a, g^b)$ ← only problematic if pairing is symmetric
- First applications of pairings was to break discrete log on certain elliptic curves
  - given (g, g^a), project into target group $e(g,g)$, $e(g,g)^a$ where discrete log may be easier to solve

New computational assumptions in pairing-based cryptography: discrete log and CDH should still hold

- Bilinear Diffie-Hellman: given g, h, g^a, g^b, distinguish $e(g,h)^{ab}$ from random
- 3-way Diffie-Hellman: given g, g^a, g^b, g^c, distinguish $e(g,g)^{abc}$ from random
- Symmetric external Diffie-Hellman (SXDH): if pairing is asymmetric, DDH holds in each base group
- k-linear assumptions: generalizations of DDH to higher rank matrices

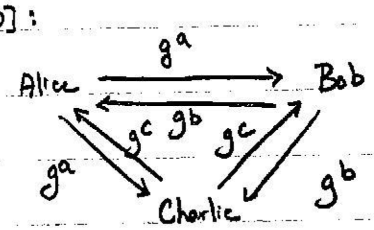
First application: 3-party non-interactive key exchange

2-party NIKE:



provably secure under DDH

3-party NIKE [Joux 00]:



shared secret : $e(g,g)^{abc}$

eavesdropper sees g^a, g^b, g^c

$\Longrightarrow$ cannot distinguish $e(g,g)^{abc}$ from random

Second application: short signatures [Boneh-Lynn-Shacham 2001]

Existing signatures (128-bits of security):

RSA signatures:	2048 bits	$\tilde{O}(\lambda^3)$
ECDSA signatures:	512 bits	4λ
Schnorr signatures:	384 bits	3λ
BLS signatures:	256 bits	2λ

$\mathrm{KeyGen}(1^\lambda) \to (vk, sk)$: $\alpha \xleftarrow{R} \mathbb{Z}_q$

$vk = (g_1, g_2, g_2^\alpha)$ $\quad sk = \alpha$

signature is just a single group element (2λ bits) $\longrightarrow$ $\mathrm{Sign}(sk, m) \to \sigma : H(m)^\alpha$ $\qquad H: m \to G_1$

$\mathrm{Verify}(vk, m, \sigma)$: check $e(\sigma, g_2) \stackrel{?}{=} e(H(m), g_2^\alpha)$

Existentially unforgeable under aCDH assumption in random oracle model:

$$\text{given } g_1, g_1^a, g_1^b, g_2, g_2^a \not\Rightarrow g_1^{ab}$$

Proof idea. Given aCDH challenge $(g_1, g_2, g_1^a, g_1^b, g_2^a)$, set verification key to be (g_1, g_2, g_2^a). For one of the random oracle queries, program output to be g_1^b. Then, successful forgery is g_1^{ab}. Signing queries handled by choosing random exponent x, setting $H(m)$ to g_1^x and computing signature as $(g_1^a)^x$.

Beyond Public-Key Encryption

Standard public-key encryption: need knowledge of public key to encrypt (public key different for each user!)

Can the public key be an arbitrary string (e.g., email address, username, etc.)

Identity-based encryption [Shamir 84]: encrypt with respect to identities

↳ major open problem and solved by Boneh-Franklin in 2001 using pairings (and concurrently by Cocks in 2001) ⟶ Start of pairings-based cryptography

More formally:

$\text{Setup}(1^\lambda) \to (\text{mpk}, \text{msk})$ ← global public parameters (mpk); master secret key (msk)

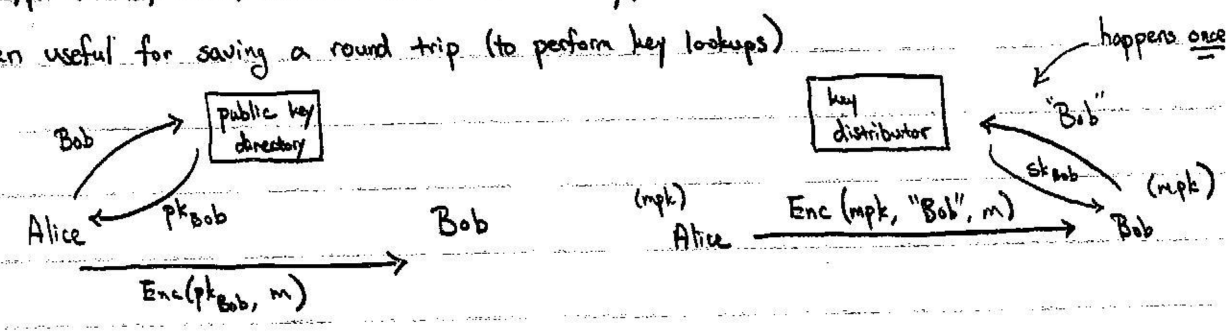
$\text{Encrypt}(\text{mpk}, \text{id}, m) \to ct_m$

$\text{KeyGen}(\text{msk}, \text{id}) \to sk_{id}$

$\text{Decrypt}(sk_{id}, ct_m) \to m$ if ct_m is encrypted to the id (and $\perp$ otherwise)

can be viewed as compressing an exponential number of public keys into mpk

Is often useful for saving a round trip (to perform key lookups)



Basic scheme (Boneh-Franklin): Asymmetric pairing-based group (of order q), $g_1 \in \mathbb{G}_1$

$\text{Setup}(1^\lambda)$: $s \xleftarrow{R} \mathbb{Z}_q$

$\text{mpk}: g_1^s = h_1 \quad \text{msk}: s$

$\text{Encrypt}(\text{mpk}, \text{id}, m)$: $r \xleftarrow{R} \mathbb{Z}_q$

$g_1^r, \; m \cdot e(h_1^r, H(\text{id}))$

ElGamal:

$s \xleftarrow{R} \mathbb{Z}_q$

$pk: g^s = h \quad sk: s$

$r \xleftarrow{R} \mathbb{Z}_q$

$g^r, \; m \cdot h^r$

Decryption: need another way to construct

$$e(h_1^r, H(\text{id})) = e(g_1, H(\text{id}))^{rs} = e(g_1^r, H(\text{id})^s)$$

Decryption: compute h^r by taking

$$(g^r)^s = (g^s)^r = h^r$$

one used with public parameters
one used with secret parameters

Key idea in pairings: exploit bilinearity and obtain two ways to compute a quantity

IBE: $\underbrace{e(g_1^r, H(id)^s)}_{\text{knowledge of secret key } H(id)^s} = e(g_1, H(id))^{rs} = \underbrace{e(g_1^s, H(id))^r}_{\text{knowledge of public key } g_1^s \text{ and randomness } r}$

BLS signatures: $\underbrace{e(H(m)^\alpha, g_2)}_{\text{requires knowing } \alpha \text{ to compute } H(m)^\alpha} = \underbrace{e(H(m), g_2^\alpha)}_{\text{all publicly computable}}$

Beyond IBE: Why stop with identities?

Attribute-based encryption: secret keys associated with policies

$\text{KeyGen}(msk, f) \to sk_f$ (policy)
$\text{Encrypt}(mpk, x, m) \to (x, ct)$ (attribute)

$\Longrightarrow$ decryption works if $f(x) = 1$

attribute is security clearance
policy is minimum level of security clearance
$\Downarrow$
encryption scheme allows access control

Predicate encryption: attributes are also hidden

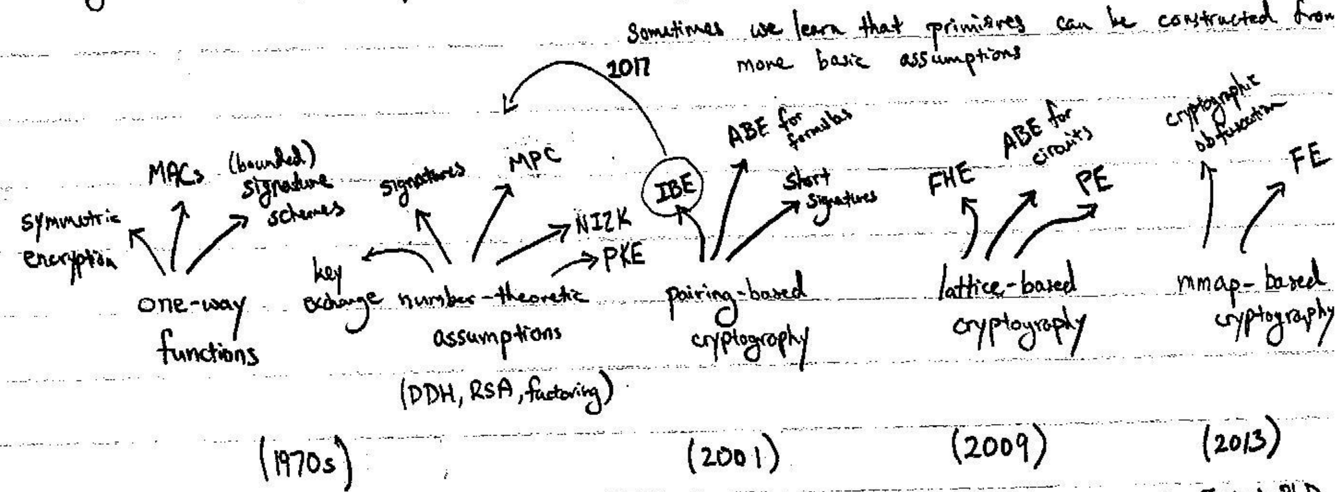
Functional encryption:

$\text{KeyGen}(msk, f)$
$\text{Encrypt}(mpk, x)$

$\Longrightarrow$ decryption outputs $f(x)$

general primitive that captures all of the existing notions (very powerful primitive!)

The Big Picture: cryptography is about identifying sources of hardness and leveraging them



... like a map? Construct trilinear map or multilinear map? Instant PhD question