Pairing-based Crypto (Lec 8, Apr 25 24)

So far, we've talked about groups $G$ of prime order $q$, where DLog is assumed to be hard (e.g. EC groups).

i.e. given uniformly random $r \in G$, it should be hard to efficiently compute $x \in \mathbb{Z}_q$ s.t. $g^x = r$. ($G = \langle g \rangle$)

Bilinear Group:

A cyclic group $G$ of order $q$, generator $g$, which also has a pairing:

A bilinear operation $e : G \times G \rightarrow G_T$ (another cyclic group of order $q'$, generator $g_T$)

that satisfies:

1) Bilinearity: For every $x, y \in \mathbb{Z}_q$,

\[ e(g^x, g^y) = e(g, g)^{xy} \in G_T \]
"Linear" in the exponent of both inputs]

Equivalently, for every \( x, y, z \in \mathbb{Z}_q \),

\[
e(g^x, gy^z) = e(g^x, g^y) \cdot e(g^x, g^z)
\]

\[= e(g, g)^{x(y+z)}
\]

Also,

\[
e(g^x+y, g^z) = e(g^x, g^z) \cdot e(g^y, g^z)
\]

2) Non-degenerate : \( e(g, g) = g^T \) is a generator of \( G_T \).

3) Efficiently computable.

Groups where DLog is hard, and additionally, there is a pairing \( e \), are \underline{VERY} useful for cryptography!

Weil (1940) came up with a pairing for some specific Elliptic curve groups.

(later variants by Tate & Ate, efficient algos by Miller)

[Note that if \( G_1 = \langle g \rangle \) is an EC group, then \( g^x = g \oplus g \oplus \ldots \) add \( x \) times,]
The above defn. is for a symmetric pairing.

**Asymmetric Pairing:**

Let $G_1, G_2$ be different groups of prime order $q$, $G_1 = \langle g_1 \rangle$, $G_2 = \langle g_2 \rangle$. Then, $e: G_1 \times G_2 \rightarrow G_T$ has the same 3 properties as defined above.

In practice, asymmetric pairings are used for groups over an elliptic curve:

* $G_1$ is a subgroup of $E(F_p)$
* $G_2$ is a subgroup of $E(F_{p^e})$
* $G_T$ is a subgroup of $F_{p^e}$ ($F_p$: finite field, $\mathbb{Z}_p$ with $+$ and $\cdot$) ($F_{p^e}$: finite field extension of $F_p$ of size $p^e$).

Additionally,
Best known log algo for $G_1, G_2$ is generic, i.e. $O(\sqrt{q})$ where $q = |G_1| = |G_2|$

An efficient pairing exists, $e: G_1 \times G_2 \rightarrow G_T$.

Note, the target group is NOT an elliptic curve group!

[Update: Apr 27] We do need log to be hard in $G_T$. Because, an algo to solve log in $G_T$ can be used to solve log for $G_1$ or $G_2$. Why?

We will mostly focus on symmetric pairings today.

What about DDH? (in the source group)

Turns out, in a group with symmetric pairing, DDH is EASY!

Consider a group $G_1 = \langle g \rangle$ of order $q$, with a pairing $e: G_1 \times G_1 \rightarrow G_T$.

Recall the DDH assumption:
\[
\{ (g^a, g^b, g^{ab}) : a, b \in \mathbb{Z}_q \} = \{ (g^a, g^b, g^c) : a, b, c \in \mathbb{Z}_q \}
\]

Algorithm to break DDH for \( G^1 \):
given a tuple \((g^a, g^b, h)\),
check if \( e(g^a, g^b) = e(g, h) \)?
if equal, \( h \) must be \( g^{ab} \). Because,
\[ e(g^a, g^b) = e(g, g)^{ab} = e(g, g^{ab}) \] by Bilinearity.
* For asymmetric pairings over \( G_1 \times G_2 \)
DDH may still be hard in \( G_1 \) or \( G_2 \) if these groups don't have symmetric pairings.

**Applications**

1. **3-party Key Exchange**: 3 parties want to agree on a key \( k \), e.g.,

**Recall**,

**Diffie-Hellman's 2-party Non-Interactive Key Exchange (NIKE)**:

[Let \( G^1 = \langle g \rangle \) be a group of order \( q \).]

\[
\begin{align*}
\text{Alice} & \quad \text{sample} \quad a \in \mathbb{Z}_q \\
\text{Bob} & \quad \text{sample} \quad b \in \mathbb{Z}_q \\
\text{Publish} \quad A = g^a & \quad \xrightarrow{A} \quad \text{Publish} \quad B = g^b
\end{align*}
\]

E.g., post on her website

\[ \]
Shared key:

compute $B^a A^b$

$$= (g^b)^a \overset{\text{Equal}}{\rightarrow} = (g^a)^b.$$ 

* Correct ✓

* Non-Interactive ✓ once Alice publishes $A$, she does NOT need to interact with anyone!!

* Secure? Key should be pseudo-random for an adversary who eaves-drop:

By DDH assumption over $G$,

$$(g^a, g^b, g^{ab}) \equiv_c (g^a, g^b, h)$$

(Leavesdropper only sees $g^a, g^b$) a $c$-element in $G$

But what about 3-party NIKE?

VI: we could generalize the above...

Round 1% $A / B$

C

Alice: $a \in \mathbb{Z}_q$

$A \leftarrow g^a$

Bob $B \leftarrow \mathbb{Z}_q$

$B \leftarrow g^b$

Charlie $C \leftarrow \mathbb{Z}_q$

$C \leftarrow g^c$
To generate a shared key: Round 2:
- Alice computes $k_{AB} = B^a$ and sends to C.
- Computes $k_{AC} = C^a$ and sends to B.
- Bob sends $k_{BC} = C^b$ to A.

Final key:

Alice: $(k_{BC})^a = (C^b)^a = (g^{gb})^a = g^{abc}$

Similarly, Bob: $(k_{AC})^b$, Charlie: $(k_{AB})^c$

Correct ✓ Secure by DDH ✓

Issue: This is INTERACTIVE! : (Godel Prize)

After Round 1:
Alice computes $e(B, C)^a = e(g^b, g^c)^a$

$= e(g^b, g)^{abc}$

Bob similarly computes $e(A, C)^b$.

Charlie " " " $e(A, B)^c$.

Shared key: $e(g, g)^{abc}$ !!!
Correct \checkmark Non-Interactive \checkmark Secure?

How do we argue that $e(g, g)^{abc}$ is pseudo-random given $A, B, C$?

**ANOTHER ASSUMPTION**
(also called Decisional-BDH)

Bilinear DDH assumption for group $G_1$:

the 2 distributions are computationally indistinguishable:

\[
\begin{pmatrix}
(g^a, g^b, g^c) \\
e(g, g)^{abc}
\end{pmatrix}
\approx_c
\begin{pmatrix}
(g^a, g^b, g^c, e(g, g)^{xy}) \\
a, b, c, y \in \mathbb{Z}_q^2
\end{pmatrix}
\]

(or no efficient adv. can distinguish b/w these distributions with non-negligible advantage.

* DBDH assumption relies on DLog

in $G_1$!!!

Q: Why?
Identity-based Encryption:

Let's say Alice wants to send an encrypted email to Bob.

- Can use Public-key Encryption:

Alice

\[ \text{pk}_A: \text{public} \]

Bob

\[ \text{pk}_B: \text{public} \]

\[ C = \text{Enc}(m, \text{pk}_B) \]

Bob can decrypt using \( \text{sk}_B \).

Now, what if Alice wants to email Carol?

- Alice also needs to know \( \text{pk}_C \).

- NEED TO STORE 1 PK PER CONTACT!!

Soln.: IBE (defined by Shamir '84)

Encrypt to "Bob" instead of \( \text{pk}_B \).

\[ \begin{align*} 
\text{Alice} & \quad \text{mpk, sk}_A \quad \text{mpk, sk}_B \\
\text{Trusted Party} & \quad \text{mpt, sk}_B \\
\text{Bob} & \end{align*} \]
Now, Alice calls $\text{Enc}(\text{mpk}, "\text{Bob}", m) = C$.

Formally, IBE has 4 PPT algorithms:

\[
\begin{align*}
\text{Setup} (1^\lambda) &\rightarrow \text{msk}, \text{mpk}. \\
\text{master secret key} &\rightarrow \text{master public key} \\
\text{KeyGen}(\text{msk}, \text{id}) &\rightarrow \text{skid}. \\
\text{e.g. skid} &\equiv \text{KeyGen}(\text{msk}, "\text{Alice}") \\
\text{Enc} (\text{mpk}, \text{id}, m) &\rightarrow C \\
\text{Dec}(\text{skid}, C) &\rightarrow m.
\end{align*}
\]

IBE should be \textit{Correct}:

Informally, Bob can decrypt any CT encrypted to "Bob".

\textbf{Secret}:

Informally, Charlie cannot decrypt CT encrypted to "Bob", or any id \# "Charlie".

\textbf{Formal Defn}:

\[
\begin{align*}
\text{Adv} &\equiv \\
\text{msk, mpk} &\leftarrow \text{Setup}(1^\lambda) \\
\text{skid} &\leftarrow \text{KeyGen} (\text{msk}, \text{id}) \\
\text{id} &\rightarrow \text{id} \\
\text{skid} &\rightarrow \text{skid}
\end{align*}
\]
\[
\text{id}^*, \text{mo}, \text{mi} \\
\]
\[
b \leftarrow \{0, 1\}^* \\
\]
\[
c \leftarrow \text{Enc}(\text{id}^*, \text{mb}) \\
\]
\[
\text{id} \\
\]
\[
\text{sk} \leftarrow \text{id} \\
\]
\[
b' \\
\]

A wins if \( b = b' \).

IBE is secure if, \( \forall \) PPT adv. \( A \),

\[
\text{Prob. that } A \text{ wins } \leq \frac{1}{2} + \text{negl}(\lambda) \\
\]

IBE from pairings: \( \text{BF'01} \)

uses a hash function \( H \) from ids to elements in \( G_1 \).

Setup(\( 1^\lambda \)): sample \( s \leftarrow \mathbb{Z}_p \),

\[
\text{msk} = s, \quad \text{mpk} = (G_1, p, g, e, h = g^s) \\
\]
KeyGen \( (mk, id) \) : \( sk_{id} = H(id)^{g} \)

Enc (\( mpk, m, id \)) : (assuming \( m \in \mathbb{G}_{T} \))
\[
(\mathbb{G}_{T}, p, g, e, h)
\]
sample \( r \in \mathbb{Z}_{p} \). Let \( u = g^{r} \)
Let \( v = e(H(id), h^{r}) \cdot m. \)
Output \( c = (u, v) \).

Dec (\( sk_{id}, c = (u, v) \)):

need to divide \( v \) by \( e(H(id), h^{r}) \).

By bilinearity,
\[
e(H(id), h^{r}) = e(H(id), h)^{r}
\]
Also, \( h = g^{g} \), so,
\[
= e(H(id), g)^{r}
\]
\[
= e(H(id)^{g}, g^{r})
\]
\[
= e(sk_{id}, u).
\]
i.e. output \( \frac{v}{e(sk_{id}, u)} \) to decrypt!
Correct \( \checkmark \) \( \text{Correct} \) (w.r.t \( e(\text{skid}, u) = e(\text{H}^{\text{skid}}, R) \))

 secure ? can be proven from 'Bilinear DDH assumption', in the Random Oracle model.
model H as a "random oracle"

3. Signatures:

Syntax:

\[
\text{KeyGen}(1^n) \rightarrow \text{sk}, \text{vk} \rightarrow \text{Public} \quad \text{secret signing key}
\]

\[
\text{Sign}(\text{sk}, m) \rightarrow \sigma
\]

\[
\text{Verify}(\text{vk}, m, \sigma) \rightarrow 0/1
\]

given \( \sigma, m \), anyone can check that Alice signed \( m \)...

Signature scheme should be
* Correct

* Unforgeable: Informally, an adv.

can't forge signature on a msg even if it has seen signatures on other messages.
BLS Signatures:

Use a hash function $H$ from msgs to elements in $G_1$. $H^*: M \rightarrow G_1$

KeyGen: $s \leftarrow \$ Z_p$

$sk \leftarrow s$, $vk \leftarrow g^s$

Sign: $(sk, m)$:

Output $\sigma \leftarrow H(m)^s$

Verify $(vk, m, \sigma)$:

Output 1 iff $e(\sigma, g) = e(H(m), vk)$

Correct? $\sigma = H(m)^s$. So,

$LHS = e(H(m)^s, g) = e(H(m), g)^s$

by Bilinearity

$= e(H(m), g^s)$

$= e(H(m), vk) = RHS$

So, signature passes verification.
Secure? Yes, assuming CDH is hard, security proven in the random oracle model.

i.e. $H$: modeled as random oracle.

Let's start with the formal definition of unforgeability:

\[
\text{Chal.} \quad \text{Adv.}
\]

[Setup Phase]:

\[
\sk, \vk \leftarrow \text{KeyGen}(1^\lambda)
\]

\[
\vk \quad \rightarrow
\]

[Query Phase]:

(Adv can query sigs on any msg it wants)

\[
\text{Chal. computes and sends.} \quad \sigma_m \leftarrow \text{Sign}(\sk, m)
\]

\[
\sigma_m \quad \rightarrow
\]
Also, because we're in the Random Oracle model, \( \text{Adv} \) can also query the ORacle for \( H(t) \) for any \( t \).

\[ \text{Hash query: } H(t) \]

\[ H(t) \]

Note, \( \text{Adv} \) can send polynomial number of \( \text{Sign}(m) \) and \( H(t) \) queries, also, they can be interleaved e.g. \( \text{Sign}(m_1), H(m_2), \text{Sign}(m_2) \) etc.

(Finally, \( \text{Adv} \) sends a forgery:)

\[ (m^*, \sigma^*) \]

\( \text{Adv} \) wins if it did not query signature on \( m^* \) before, AND,

\[ \text{Verify}(vk, m^*, \sigma^*) = 1. \quad (\text{i.e. valid forgery}) \]

The **CDH assumption**: For group \( G_1 = \langle g \rangle \) of prime order \( q \),

\( \text{CDH is hard if it is hard to compute } g^{xy} \text{ given } g^x, g^y \text{ for } x, y \in \mathbb{Z}_q \).
Formally,

$$\Pr \left[ A \left( g^x, g^y, g^t, g^g \right) = g^{xy} \; : \; x, y \in \mathbb{Z}_q \right] \leq \text{negl}(\lambda)$$

We will prove security of BLS by a reduction to CDH.

I.e., an Adv that breaks BLS Sig. can be used to break CDH.

(Since CDH is assumed to be hard, no Adv can break BLS unforgeability.)

Given A : Adv that breaks BLS security, we’ll build B, that breaks CDH.

CDH

\[ \text{Adv} \]

\[ \text{B} \]  
(acts as the Adv for \( A \)'s game)
CDH: \( h_1 = g^x, h_2 = g^y \) \( \rightarrow \) \( B \)

**Main idea:** \( B \) wants to compute \( g^{xy} \). We can think of \( g^{xy} \) as \((g^x)^y = g^{xy}\). This looks like a signature for \( s = y \).

For some msg \( m \) s.t. \( H(m) = g^x \).

Hopefully, forgery \( s^x = g^{xy} \) but now how can \( B \) enforce that Hash of \( m^x \) is \( g^x \) ???

This is what \( B \) does:

1) \( B \) sets \( vK = g^y \), and sends it to \( A \)

(we want \( s = y \), so \( vK = g^y \), but \( B \) doesn't know \( y \)).

Now, \( A \) will query \( B \) for signatures and Hashes of arbitrary msgs.

**Informal Idea:** Let's say \( A \) queries \( \text{Sign}(m) \).

\( B \) needs to respond with a valid signature, i.e. \( H(m)^y \).
But, B does NOT know y, so HOW can it compute \( H(y) \)?

* B will program the Random Oracle!!

Here's how B programs \( H \):

**Step 2.**

Let's say, \( Q_R \) is the number of queries that A does to \( H() \). \( Q_R \) must be \( \text{poly}(\lambda) \) bc A is PPT.

(We will index Hash queries as \( 1, 2, \ldots, Q_R \))

\( B \) guesses an index \( j^* \in \{0, 1\}^{Q_R} \).

**Step 3.** Here's how \( B \) responds to \( H(t) \) query by \( A \):

Let's say this is the \( j \)th query to \( H \).

- If \( j \neq j^* \): \( B \) samples \( y_t \in \mathbb{Z}_q \) and replies with \( g^{y_t} \).
- \( B \) also stores \( (j, t, y_t) \) in a map \( M \).

- If \( j = j^* \): \( B \) replies with \( h_t = g^{x^j} \).
(note how B now knows the Dlog of $H(t)$ for all queries except the $j^{th}$ one.
So, B can compute $\text{Sign}(t)$ as $h_2 = H(t)$)

Also note, that B's responses to the $H$ queries are indistinguishable from uniform random, be
$y_t$ sampled uniformly randomly
$\Rightarrow y_t$ is uniformly random,
and $h_t$ is also uniform random be the CDA real. samples $x$ uniform randomly from $Z_q$

---

step 4.) Here's how B responds to $\text{Sign}(m)$ queries by A:
(A1.) Let's assume that A always queries $H(m)$ before querying $\text{Sign}(m)$, and all queries to $H$ are unique.

Then, B checks its map $M$ to find a tuple $(im, m, y_m)$ if $m$ was the $j^{th}$ $H$ query.
Otherwise, $y_m$
B replies with $(h_2)$. 

Otherwise, $y_m$
B replies with $(h_2)$. 

If no such tuple exists, B aborts.
This is a valid signature on \( m \) bc,

we know that B set \( H(m) = g^{ym} \), so,

\[
\sigma_m = H(m) = (g^{ym})^s = (g^y)^{ym} = b_2
\]

So, B acts just like a real BLS challenger!

Eventually, A sends a forgery \((m^*, \sigma^*)\) to B.

(Let's assume that A queries \( H(m^*) \) before sending the forgery.

If \( m^* \) was NOT the \( j^\text{th} \) Hash query, then B aborts.

Otherwise,

B simply outputs \( \sigma^* \).

Claim: If A wins its game, and

B does NOT abort, then, B breaks CDH.

Why?

If A wins: Verify \((vk, m^*, \sigma^*) = 1 \)

\[ i.e, \quad e(\sigma^*, g) = e(H(m^*), vk) \]

\[ = e( g^{ym^*}, g^{yn^*}) = e( g^{ym^*}, g) \]

\( H(m^*) \) was set to \( g^y \) by B, \( \Rightarrow m^* \) was \( j^\text{th} \) query

and \( vault \) " " " " \( g^{yn^*} \) by B.)
Exactly what B wants!!

What's the probability that B aborts?

A has OR queries to H:

\[ t_1, t_2, \ldots, t_{QR} \]

By assumption A2, \( m^* \) must be one of these queries; let's say it's the \( j \)-th query.

\[
\text{Prob that } B \text{ guesses } j \text{ correctly} = \text{Prob that } j^* = j = \frac{1}{QR}.
\]

(\( B \) uniformly samples \( j^* \) from [\( QR \)])

Claim 1: \( \text{Prob } (B \text{ doesn't abort }) = \frac{1}{QR} \)

Why? If \( B \) guesses \( j \) correctly, then, \( j^* = j \).

Then, \( m^* \) was the \( j \)-th query. Since \( A \) cannot query sign on \( m^* \), \( B \) will not abort on any \( \text{Sign}(m) \) query.
Also since \( m^* \) is the \( j \)th query, B will not abort in the end.

So, if A breaks BLS with prob \( \epsilon \), then B breaks CDH with probability:

\[
\Pr(\text{A wins and B doesn't abort})
\]

\[
= \Pr(\text{B doesn't abort}) \times \Pr(\text{A wins/ B doesn't abort})
\]

\[
\downarrow \\
\uparrow \\
\Pr(\text{B wins}) = \epsilon \times \frac{1}{O\epsilon}
\]

Also, B is efficient.

So if \( \epsilon \) were non-negligible, B's advantage would be non-negligible, which would break our assumption that CDH is hard.

Assumptions A1, A2 are easy to remove, see Dan & Shoup's book (Ch. 15.5).
Programming the Random Oracle:

Notice that in the proof, B chose how to reply to each Hash query (e.g., $g^x$ or $h^2$). This only makes sense in the Random Oracle model. (B’s replies are uniformly random.) In the real world, e.g., $H = SHA3$, B cannot set $H(t)$ to whatever, $H(t)$ must be $SHA3(t)$. So this proof does not work in the standard model (i.e., real).

But, programmable ROs are widely used in security proofs, and schemes like BLS are also widely used (and haven’t been broken...)

Aggregating BLS Signatures:

One of the main reasons why BLS...
Signatures are amazing is that BLS sigs are easy to aggregate:

* Can do non-interactive threshold signing: e.g. any 3 out of 5 parties can sign a msg.
* Can aggregate BLS signatures by multiple parties (on same or different messages) non-interactively!!

E.g. Alice: Bob:

\[ \text{sk}_A, \text{vk}_A \leftarrow \text{KGen}() \quad \text{sk}_B, \text{vk}_B \leftarrow \text{KGen()} \]

\[ \sigma_A = \text{Sign}(\text{sk}_A, m) \quad \sigma_B = \text{Sign}(\text{sk}_B, m) \]

\[ = \text{H}(m)^{\text{sk}_A} \quad = \text{H}(m)^{\text{sk}_B} \]

To aggregate:

Aggregate Sig: \[ \sigma = \sigma_A \times \sigma_B = \text{H}(m) \]

Aggregate PK: \[ \text{vk} = \text{vk}_A \times \text{vk}_B = g^{\text{sk}_A + \text{sk}_B} \]
To verify, just check if
\[ e(\sigma, g) = e(H(m), \nu_k) \? \]

Note, the scheme above is NOT actually secure, due to the
ROGUE KEY ATTACK.

But, there are many efficient ways
to make it secure,
(e.g. BGN eprint 2018/483,
proof of possession, etc.)