Pairing-based Crypto (Lec 8, Apr 22'26)

- Bilinear groups.
- Applications in Crypto
  - 3-party Key Exchange
  - BLS Signatures
    - Defining Signatures
    - The BLS construction
    - Security proof in ROM !!!

(- Identity Based Encryption : In Lecture Notes.)

So far, we've talked about groups G_1 of prime order q, where DLog is assumed to be hard. (e.g. EC groups)

i.e. given uniformly random $h \in G_1$, it should be hard to efficiently compute $x \in \mathbb{Z}_q$ s.t. $g^x = h$. ($G_1 = \langle g \rangle$)

Bilinear Groups:

a cyclic group G_1 of order q (prime), generator g, which also has a **pairing**:

A bilinear operation $e : G_1 \times G_1 \to G_T$

(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 $x, y, z \in \mathbb{Z}_q$,

$$e(g^x, g^{y+z}) = e(g^x, g^y) \cdot e(g^x, g^z)$$
$$= e(g, g)^{x(y+z)}$$

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 <u>VERY</u> 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 $g^x = g \boxplus g \boxplus \ldots$: add x times,

[(⊞ was defined in last class)]

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, $e: G_1 \times G_2 \to G_T$.

(G_1, G_2: SOURCE GROUPS; G_T: TARGET GROUP)

has the same 3 properties as defined above.

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

* G_1 is a (prime-order) subgroup of $E(\mathbb{F}_p)$
* G_2 is a subgroup of $E(\mathbb{F}_{p^e})$
* G_T is a subgroup of $\mathbb{F}_{p^e}$

($\mathbb{F}_p$: finite field, $\mathbb{Z}_p$ with + and *

$\mathbb{F}_{p^e}$: finite field extension of $\mathbb{F}_p$ of size p^e.)

Additionally,

* Best known DLog algo for G_1, G_2 is generic,

i.e. $\tilde{O}(\sqrt{q})$ where $q = |G_1| = |G_2|$

* An efficient pairing exists;

$e : G_1 \times G_2 \to G_T$.

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

(Note: We do need DLog to be hard in G_T. Because, an algo. to solve DLog in G_T can be used to solve DLog for G_1 or G_2. D. why?)

We will mostly focus on symmetric pairings today

* What about DDH? (in the source groups)

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

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

Recall the DDH assumption:

$\{(g^a, g^b, g^{ab}) : a, b \xleftarrow{\$} \mathbb{Z}_q\} \approx_c$

$\{(g^a, g^b, g^c) : a, b, c \xleftarrow{\$} \mathbb{Z}_q\}$

Algo 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 dont have symmetric pairings.

Applications :-

(1.) 3-party Key Exchange : for symmetric encryption. → 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.]

Alice		Bob
sample $a \xleftarrow{\$} \mathbb{Z}_q$		sample $b \xleftarrow{\$} \mathbb{Z}_q$
Publish $A = g^a$	$\xrightarrow{A}$	Publish $B = g^b$.
	$\xleftarrow{B}$	

e.g. Post on her website

Shared key:

Compute B^a $\qquad$ A^b

$= (g^b)^a$ $\quad \xleftrightarrow{\text{Equal}} \quad$ $= (g^a)^b$.

* 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:

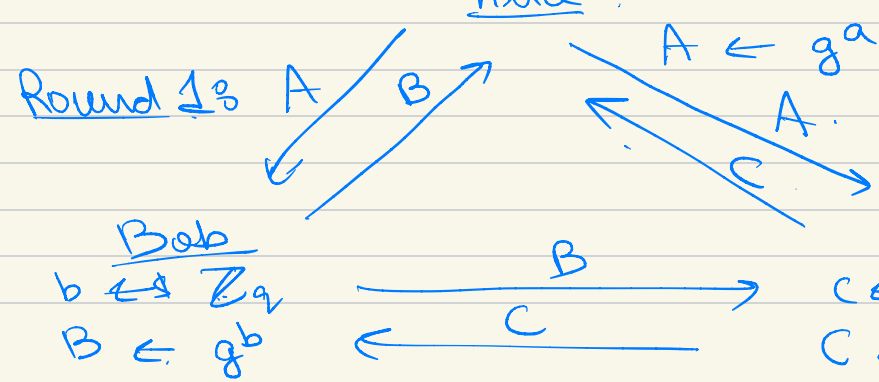
By DDH assumption over G,

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

(Eavesdropper only sees g^a, g^b) $\qquad$ u.r. element in G

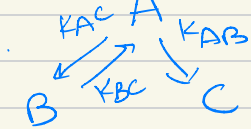
But what about 3-party NIKE?

V1: we could generalize the above...



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: $(k_{BC})^a = (C^b)^a = (g^c)^{ab} = g^{abc}$

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

Correct ✓ Secure by DDH ✓

Issue: This is INTERACTIVE! ☹

Joux '04 (Godel Prize)

After Round 1,

Alice computes $e(B, C)^a = e(g^b, g^c)^a$

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

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

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

Shared Key: $e(g, g)^{abc}$!!!

Correct ✓ Non-Interactive ✓

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

ANOTHER ASSUMPTION! (also called Decisional-BDH) ← VERY commonly used!

Bilinear DDH assumption for group G_T:

the 2 distributions are computationally indistinguishable:

$$\left\{ \left(g^a, g^b, g^c, e(g,g)^{abc} \right) : a, b, c \leftarrow\$ \mathbb{Z}_q \right\}$$

$$\approx_c$$

$$\left\{ \left(g^a, g^b, g^c, e(g,g)^{y} \right) : a, b, c, y \leftarrow\$ \mathbb{Z}_q \right\}$$

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

* DBDH assumption relies on DLog in G_T !!!
Q. why?

(SKIPPED in CLASS)

(2) Identity-based Encryption:

Lets say, Alice wants to send an encrypted email to Bob:

⇒ Can use Public-Key Encryption:

Alice: pk_A : public

Bob: pk_B : public

$C = Enc(m, pk_B)$ → Bob can decrypt using sk_B.

Now, what if Alice wants to email carol?

⇒ Alice also needs to know pk_C.

⇒ NEED TO STORE 1 pk (large) PER contact!!

Soln: IBE (defined by Shamir '84)

Encrypt to "Bob" instead of pk_B.

Trusted Party.

↙ mpk, sk_A → Alice

↘ mpk, sk_B → Bob

Now, Alice calls $Enc(mpk, \text{"Bob"}, m) = C$.

Formally, IBE has 4 PPT algorithms :-
(Probabilistic Poly-Time)

1.) $Setup(1^\lambda) \rightarrow msk, mpk$.
msk → master secret key; mpk → master public key

2.) $KeyGen(msk, id) \rightarrow SK_{id}$.
e.g. $SK_A = KeyGen(msk, \text{"Alice"})$.

3.) $Enc(mpk, id, m) \rightarrow C$

4.) $Dec(SK_{id}, C) \rightarrow m$.

IBE should be * correct :
Informally,
Bob can decrypt any CT encrypted to "Bob".

Secure : Informally,

Charlie cannot decrypt CT encrypted to "Bob", or any $id \neq$ "Charlie".

Formal Defn :

Chal. | Adv.

$msk, mpk \leftarrow Setup(1^\lambda)$

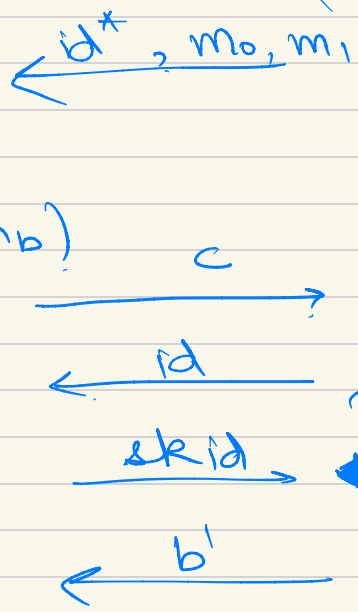
$mpk \longrightarrow$

$\leftarrow id$

$SK_{id} \leftarrow KeyGen(msk, id)$

$SK_{id} \longrightarrow$

(e.g. id = Carol: Carol is corrupt)



A wins if $b = b'$.

IBE is secure, if, $\forall$ PPT adv. A,

Prob. that A wins is $\leq \frac{1}{2} + negl(\lambda)$.

Gödel Prize! Boneh-Franklin

IBE from pairings: (BF'01)

uses a hash function H: from ids to elements in G_1.

Setup(1^λ): sample $s \leftarrow\$ \mathbb{Z}_p$.

$msk = s$, $mpk = (G_1, p, g, e, h = g^s)$

KeyGen(msk, id) : $sk_{id} = H(id)^s$

Enc(mpk, m, id) : (assuming $m \in G_T$)

(G, p, g, e, h)

Sample $r \xleftarrow{\$} \mathbb{Z}_p$. Let $u = g^r$

Let $v = e(H(id), h^r) * m$.

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, $h = g^s$, so,

$= e(H(id), g)^{rs}$

$= e(H(id)^s, g^r)$

$= e(sk_{id}, u)$.

i.e. output $\frac{v}{e(sk_{id}, u)}$ to decrypt!

Correct ✓ ($\because e(sk_{id}, u) = e(H(id), h^{r})$)

Secure ? Can be proven from Bilinear DDH assumption, in the Random Oracle model.

↓

model H as a "random oracle"

3. Signatures:

Syntax:

KeyGen(1^n) → sk, vk → Public verification key

↓

secret signing key

Sign(sk, m) → σ ∴ may not be deterministic.

Verify(vk, m, σ) → 0/1

↳ given σ, m, anyone can check that Alice signed m...

Signature scheme should be

* Correct : for all messages m, Verify outputs 1 for σ = Sign(sk, m).
* Unforgeable : Informally, an adv. can't forge signature on a msg, even if it has seen signatures on other messages.

- Boneh, Lynn, Shacham.

BLS Signatures:

Uses a hash function H: from msgs to elements in G_1.

$H: M \to G_1$

(also sample $G_1, p, g \leftarrow\$ \text{Setup}(1^\lambda)$)

KeyGen: $s \leftarrow\$ \mathbb{Z}_p$.

$sk \leftarrow s$, $vk \leftarrow g^s$ (* Similar to BF'01 IBE above.)

Sign$(sk = s, m)$:

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

Verify(vk, m, σ):

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.

Lets start with the formal definition of unforgeability :-

Chal. Adv

[Setup Phase] :

$SK, VK \leftarrow\$ KeyGen(1^\lambda)$

$VK \longrightarrow$

[Query Phase] :

(Adv can query sigs on any msg it wants)

$\longleftarrow Sign(m)$

Chal. computes and sends :

$\sigma_m \leftarrow Sign(SK, m)$

$\sigma_m \longrightarrow$

(Also, because we're in the Random oracle model, Adv can also query the Chal. for $H(t)$ for any t.)

Hash query: $H(t)$ ←

$H(t)$ →

[Note, Adv can send polynomial number of Sign(m) and $H(t)$ queries, also, they can be interleaved
e.g. Sign(m_1), $H(m_2)$, Sign(m_2) etc.]

(Finally, Adv sends a forgery:)

(m^*, σ^*) ←

Adv wins if it did not query signature on m^* before, AND,

Verify(vk, m^*, σ^*) $= 1$. (i.e. valid forgery)

The CDH assumption: For group $G = \langle g \rangle$ of prime order q,

CDH is hard if it's hard to compute g^{xy} given g^x, g^y for $x, y \xleftarrow{\$} \mathbb{Z}_q$

Formally,

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

(note: CDH is a stronger assumption than DDH)

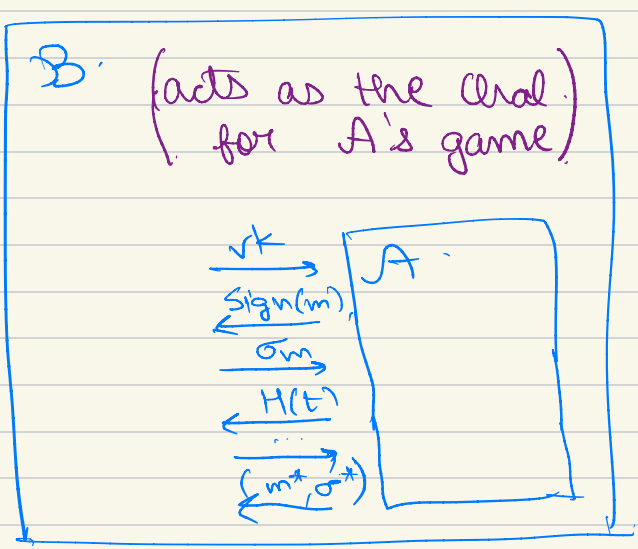
we will prove security of BLS by a reduction to CDH.

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 chal. ⟷



CDH Oral. $h_1 = g^x$, $h_2 = g^y$ → B.

MAin idea: B wants to compute g^{xy}.

we can think of g^{xy} as $(g^x)^y$ ⇒

This looks like a signature ..... for $s = y$ and for some msg m s.t. $H(m) = g^x$...

Hopefully, forgery $\sigma^* = g^{xy}$ ...

But how can B enforce that Hash of m^* is g^x ??? .

This is what B does :-

step 1.) B sets $vk = g^{y} = h_2$, 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: Lets say A queries Sign(m).

B needs to respond with a valid signature, i.e. $H(m)^y$.

But, $\mathcal{B}$ does NOT know y, so HOW can it compute $H(m)^y$?

⋆⋆ $\mathcal{B}$ will program the Random Oracle

Step 2)

Say, $\mathcal{A}$ queries $\mathcal{B}$ for $H(m)$:

$\mathcal{B}$ samples $y_m \xleftarrow{\$} \mathbb{Z}_q$, and sets $H(m) = g^{y_m}$.

$\mathcal{B}$ also stores y_m in a map M:

$M[m] = y_m$.

This means that $\mathcal{B}$ knows DLog of $H(m)$ for any m queried. So, $\mathcal{B}$ will be able to compute a signature on m as $h_2^{y_m} = H(m)^y$

Also note, that $\mathcal{B}$'s responses to the H queries are indistinguishable from uniform random, bc y_t sampled uniformly randomly $\Rightarrow$ g^{y_t} is uniformly random.

Step 3: Here's how B responds to Sign(m) queries by A:

(A1.)
* Lets assume that A always queries H(m) before querying Sign(m), and all queries to H are unique.

A1 ⇒ A must've queried H(m) before.

So, B checks the map M, to get $y_m = M[m]$, and outputs $h_2^{y_m}$.

(This is a valid signature:
$h_2^{y_m} = g^{y \cdot y_m} = H(m)^y$ ✓)

Eventually, A replies with a forgery (m^*, σ^*)., where $\sigma^* = H(m^*)^y$.

But Recall, B needs to compute g^{xy} to win its game,

i.e. we want $H(m^*)^y = g^{xy}$

⇒ we want $H(m^*)$ to equal g^x.

But HOW?

(A2).

* Lets assume that A queries $H(m^*)$ at some point, before outputting the forgery.

If we somehow knew which query to H corresponds to m^* B could just program $H(m^*)$ to be g^x (Then, forgery σ^* would be the CDH solution!)

But ofcourse, we don't know which query is for m^*...

Sol'n: Since A is PPT, it can only make poly(λ) # Queries !!!

Say, it makes Q queries to H.

We know, one of these must be for m^*, (we don't know which).

But, we can just guess which one!

i.e. B can guess $j^* \leftarrow\$ [Q]$, and act as if j^* th H query is for m^*. Informally, if B guesses

correctly and A gives a valid forgery,

↓ B wins its game! ↓

Happens with prob. $\frac{1}{Q}$ Prob = A's advantage.

So, B's advantage = $\frac{1}{Q}$. A's advantage.

Step 2.) Here's the full formal reduction.

Lets say, Q_R is the number of queries that A does to H(). Q_R must be poly(λ) bc A is PPT.

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

B guesses an index $j^* \leftarrow\$ [Q_R]$.

Step 3.) Here's how B responds to H(m) query by A:

Lets say this is the jth query to H.

if $j \neq j^*$: B samples $y_m \leftarrow\$ \mathbb{Z}_q$.
and replies with g^{y_m}
B also stores (j, m, y_m) in a map M.

if $j = j^*$: B replies with $h_1 = g^x$.

note how B now knows the DLog of $H(t)$ for all queries except the j^*th one. So, B can compute sign(t) as $h_2^{y_t} = H(t)^{y}$

Also note, that B's responses to the H queries are indistinguishable from uniform random, bc y_t sampled uniformly randomly $\Rightarrow$ g^{y_t} is uniformly random, and h_1 is also unif. random bc the CDH chal. samples x unif. randomly from $\mathbb{Z}_q$

step 4.) Here's how B responds to Sign(m) queries by A:

B checks its map M to find a tuple (i_m, m, y_m). → if m was the j^*th H query...

If no such tuple exists, B aborts.

Otherwise, B replies with $(h_2)^{y_m}$.

This is a valid signature on m bc, we know that $\mathcal{B}$ set $H(m) = g^{y_m}$, so,

$$\sigma_m = H(m)^s = (g^{y_m})^y = (g^y)^{y_m} = h_2^{y_m}.$$

So, $\mathcal{B}$ acts just like a real BLS challenger!

Eventually, $\mathcal{A}$ sends a forgery (m^*, σ^*) to $\mathcal{B}$.

By assumption A_2, we know $\mathcal{A}$ queried $H(m^*)$ at some point...

If m^* was NOT the j^* th Hash query, then $\mathcal{B}$ aborts.

otherwise,

$\mathcal{B}$ simply outputs σ^*.

Claim: If $\mathcal{A}$ wins its game, and $\mathcal{B}$ does NOT abort, then, $\mathcal{B}$ breaks CDH.

why?

If $\mathcal{A}$ wins: $\text{Verify}(vk, m^*, \sigma^*) = 1$.

i.e $e(\sigma^*, g) = e(H(m^*), vk)$

$= e(g^y, g^x) = e(g^{xy}, g)$

($H(m^*)$ was set to g^y by $\mathcal{B}$, $\because$ $\mathcal{B}$ didnt abort $\Rightarrow m^*$ was j^*th query.

and vk " " " g^x by $\mathcal{B}$)

$\Rightarrow$ $\sigma^* = g^{xy}$!!! EXACTly what B wants !! ☺

Whats the probability that B aborts?

A has Q_R queries to H:

$t_1, t_2, \ldots, t_{Q_R}$

By assumption A2, m^* must be one of these queries; lets say its the $\hat{j}$ th query.

Prob. that B guesses $\hat{j}$ correctly

$=$ Prob. that $j^* = \hat{j} = \frac{1}{Q_R}$

(be B uniformly samples j^* from $[Q_R]$)

Claim: Prob (B doesn't abort) $= \frac{1}{Q_R}$.

Why?

If B guesses $\hat{j}$ correctly, then, $j^* = \hat{j}$.

Then, m^* was the j^* th query. Since A cannot query sign on m^*, B will not abort on any Sign(m_i) query.

Also since m^* is the j^*th query,
B will not abort in the end.

So, if A breaks BLS with probab. ϵ,
then B breaks CDH with
probability = Pr(A wins and B doesn't abort)

= Pr(B doesn't abort) * Pr(A wins / B doesn't abort)

Pr(B doesn't abort) $= \frac{1}{Q_R}$.

Pr(A wins / B doesn't abort) $= \epsilon$ because B acts as a real BLS Chal., so A wins with prob. ϵ.

i.e. $Pr(B \text{ wins}) = \epsilon * \frac{1}{Q_R}$.

Also, B is efficient.

So if ϵ 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, $\mathcal{B}$ chose how to reply to each Hash query: e.g. g^{kt} or h_2. This only makes sense in the Random Oracle model. ($\mathcal{B}$'s replies are uniformly random). In the real world, e.g. $H = SHA3$, $\mathcal{B}$ 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 world).

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 :

$sk_A, vk_A \leftarrow\$ KGen()$ $sk_B, vk_B \leftarrow\$ KGen()$

$\sigma_A = Sign(sk_A, m) = H(m)^{s_A}$, $\sigma_B = Sign(sk_B, m) = H(m)^{s_B}$

To aggregate :

Aggregate Sig : $\sigma = \sigma_A * \sigma_B = H(m)^{s_A + s_B}$

Aggregate pk : $vk = vk_A * vk_B = g^{s_A + s_B}$

So,

To verify that both A and B signed m, we only need to check:

$$e(\sigma, g) = e(H(m), vk) \; ?$$

* 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. BDN eprint 2018/483, Proof of possession, etc.)