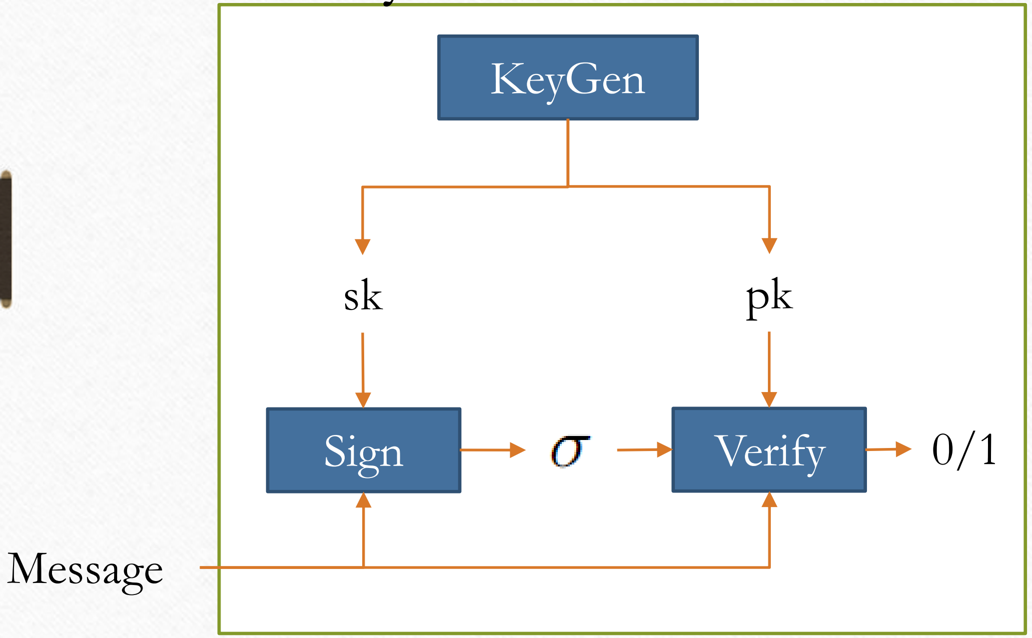
Hardening Signature Schemes via Derive-then-Derandomize: Stronger Security Proofs for EdDSA

Mihir Bellare, **Hannah Davis**, Zijing Di

PKC May 2023

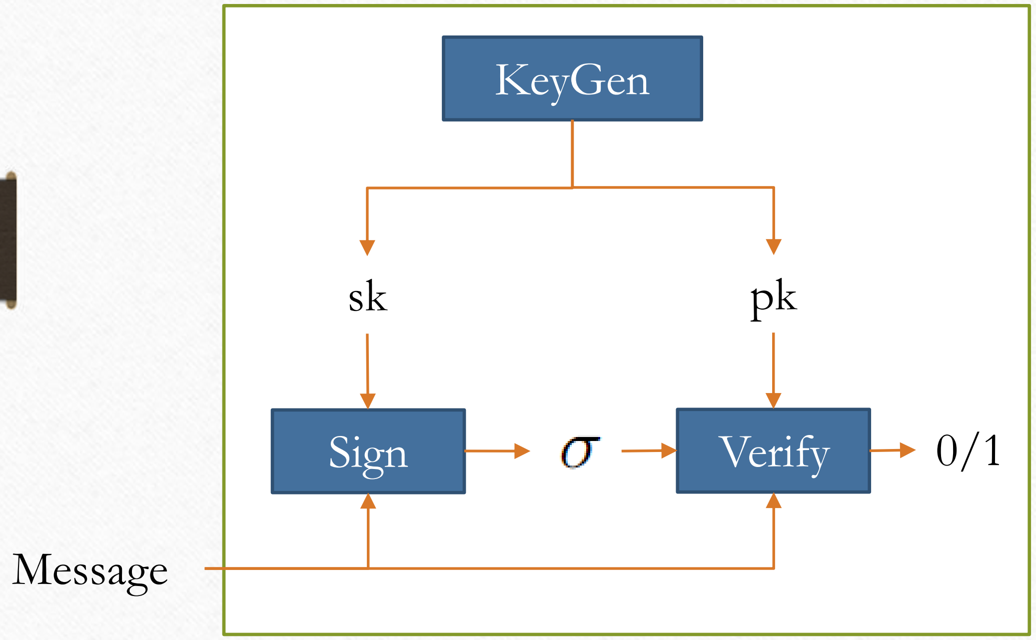
Digital Signatures

Syntax Definition



Digital Signatures

Syntax Definition



KeyGen

sk

pk

Sign

σ

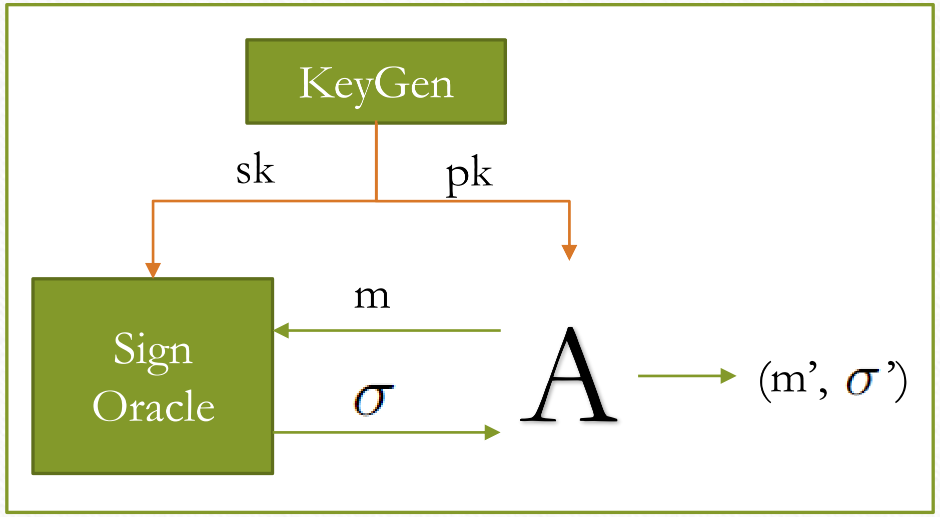
Verify

0/1

Message

Security Definition

The **UF-CMA** advantage of an adversary **A** attacking a scheme **S** is the probability that **A** produces a valid signature on any unsigned message



KeyGen

sk

pk

m

Sign Oracle

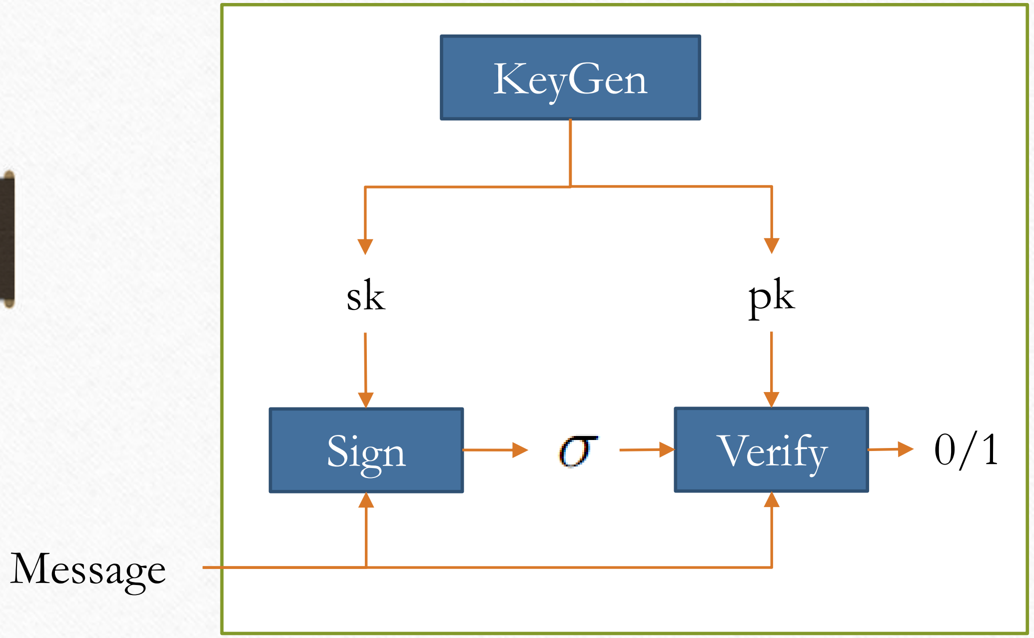
σ

A

(m', σ')

Schnorr Signatures[Schnorr91]

Syntax Definition



Order-p
DL Group
g

+

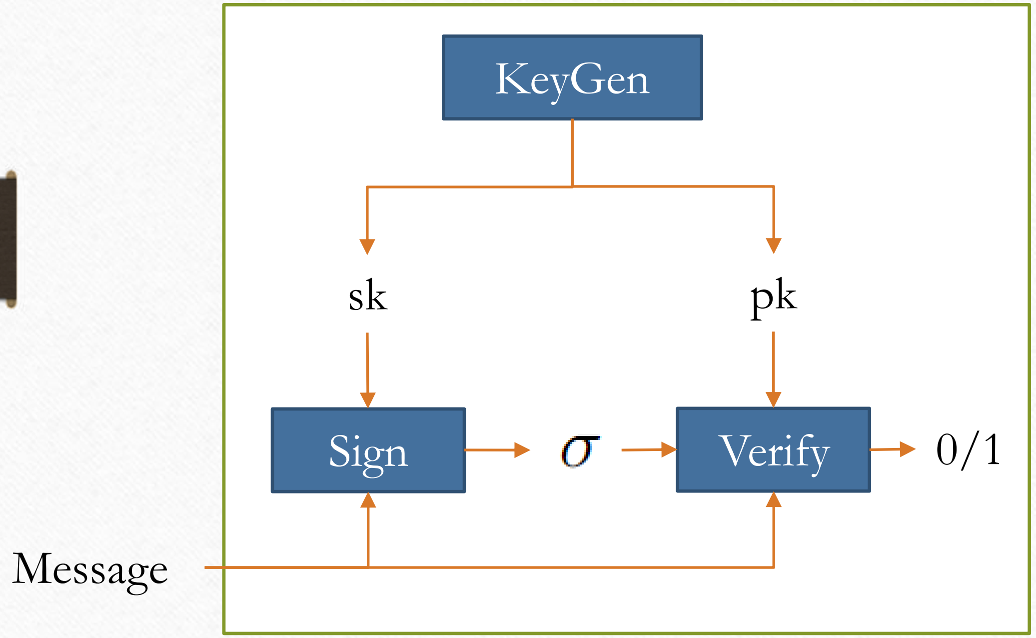
Hash function
H

Discrete Log (DL) problem:

Given generator g and random group element R compute r such that $R = r \cdot g$

Schnorr Signatures [Schnorr91]

Syntax Definition



Order-p
DL Group
g

+

Hash function
H

KeyGen:

```
$sk \leftarrow_{\$} \mathbb{Z}_p$
$pk \leftarrow sk \cdot g$
Return $(sk, pk)$
```

Sign[H](sk, pk, M):

```
$r \leftarrow_{\$} \mathbb{Z}_p$ ; $R \leftarrow r \cdot g$
$c \leftarrow H(R \| pk \| M)$
$z \leftarrow (sk \cdot c + r)$ mod p
Return $(R, z)$
```

Verify[H](pk, M, σ):

```
$(R, z) \leftarrow \sigma$
$c \leftarrow H(R \| pk \| M)$
Return $[[z \cdot g == c \cdot pk + R]]$
```

Schnorr Signatures[Schnorr91]

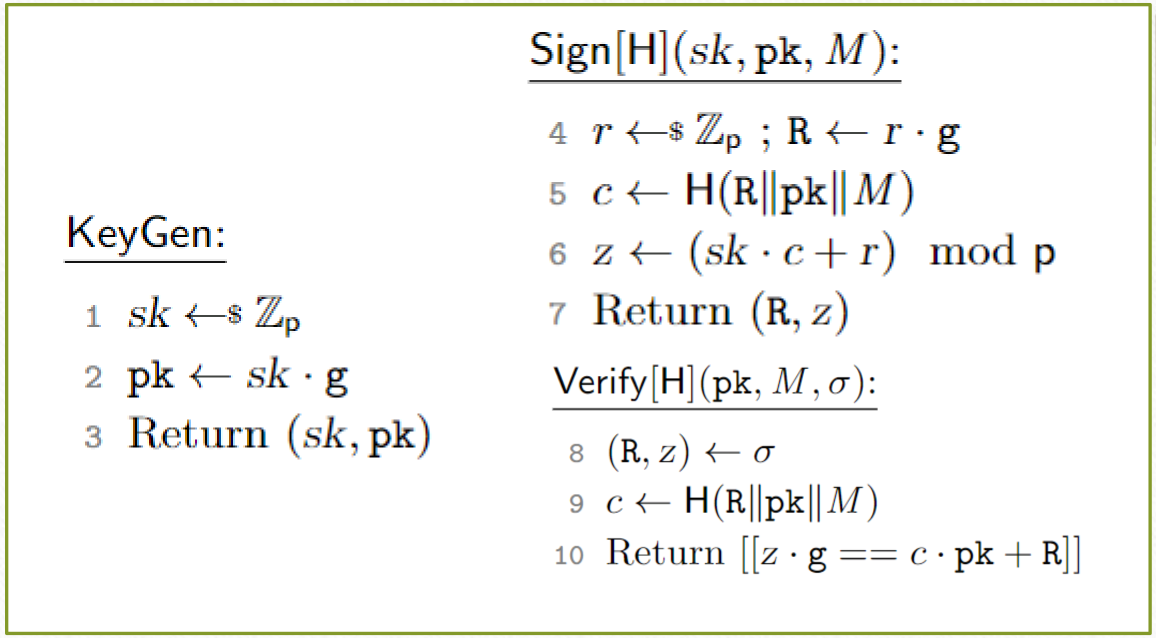
Pros

- Simple
- Efficient (for a DL-based scheme)
- Short signatures compared to RSA

Order-p DL Group
g

+

Hash function
H



```
KeyGen:
    $sk \leftarrow_{\$} \mathbb{Z}_p$
    $pk \leftarrow sk \cdot g$
    Return $(sk, pk)$

Sign[H]$(sk, pk, M)$:
    $r \leftarrow_{\$} \mathbb{Z}_p$ ; $R \leftarrow r \cdot g$
    $c \leftarrow H(R \| pk \| M)$
    $z \leftarrow (sk \cdot c + r) \mod p$
    Return $(R, z)$

Verify[H]$(pk, M, \sigma)$:
    $(R, z) \leftarrow \sigma$
    $c \leftarrow H(R \| pk \| M)$
    Return $[[z \cdot g == c \cdot pk + R]]$
```

Schnorr Signatures[Schnorr91]

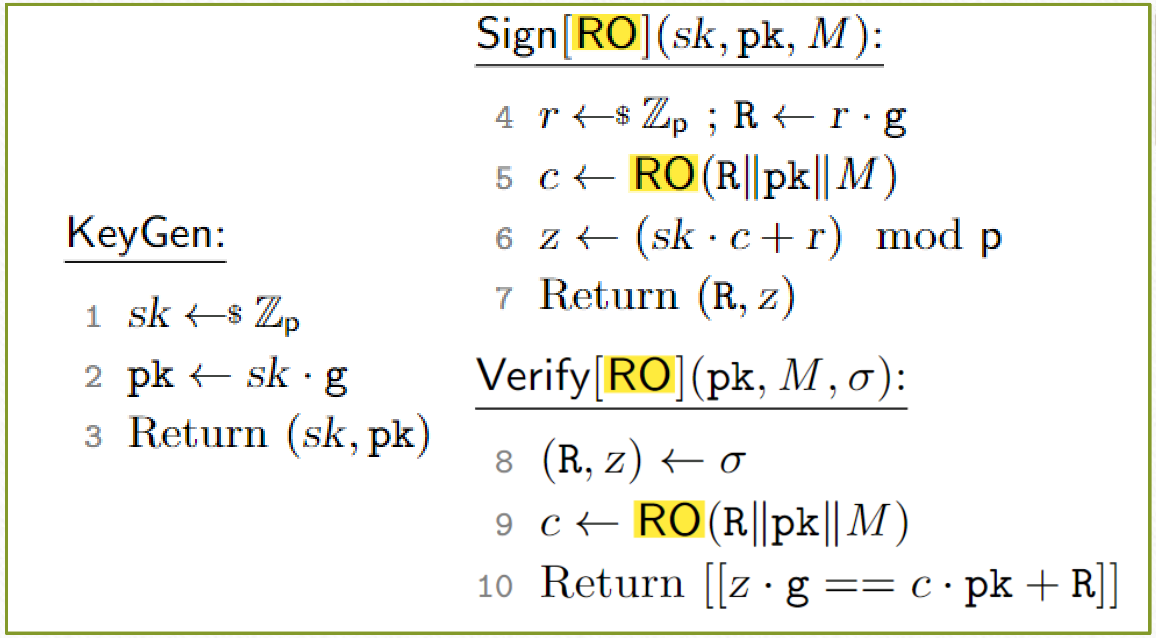
Pros

- Simple
- Efficient (for a DL-based scheme)
- Short signatures
- Reducible to DL in the ROM

Order-p DL Group
g

\+

Random oracle
RO

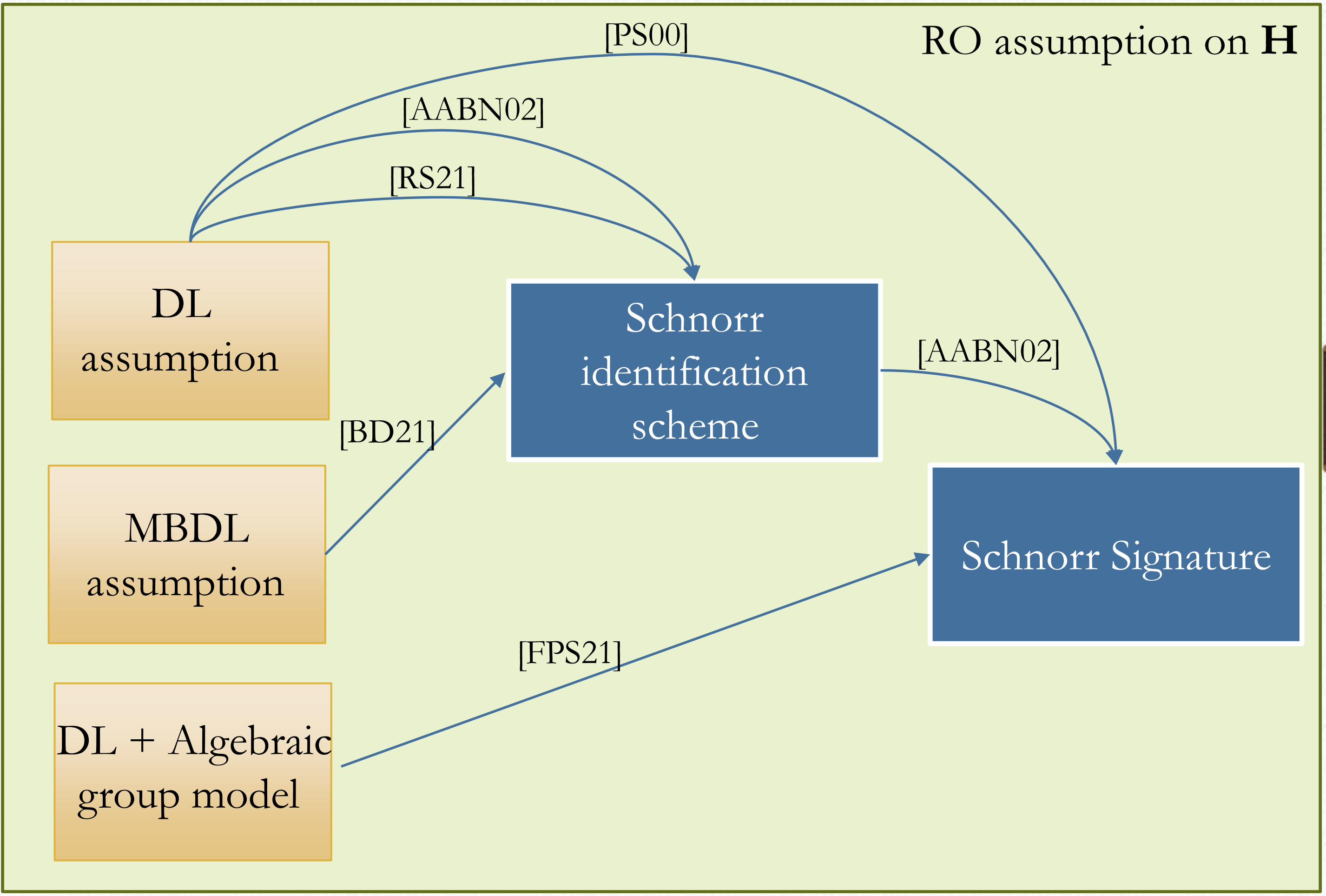


```
KeyGen:
    $sk \leftarrow_{\$} \mathbb{Z}_p$
    $pk \leftarrow sk \cdot g$
    Return $(sk, pk)$

Sign[RO]($sk, pk, M$):
    $r \leftarrow_{\$} \mathbb{Z}_p$ ; $R \leftarrow r \cdot g$
    $c \leftarrow$ RO($R \| pk \| M$)
    $z \leftarrow (sk \cdot c + r) \bmod p$
    Return $(R, z)$

Verify[RO]($pk, M, \sigma$):
    $(R, z) \leftarrow \sigma$
    $c \leftarrow$ RO($R \| pk \| M$)
    Return $[[z \cdot g == c \cdot pk + R]]$
```

Schnorr Signatures [Schnorr91]

Pros

- Simple
- Efficient (for a DL-based scheme)
- Short signatures
- Reducible to DL in the ROM
- Many formal security proofs with varying tightness & starting assumptions



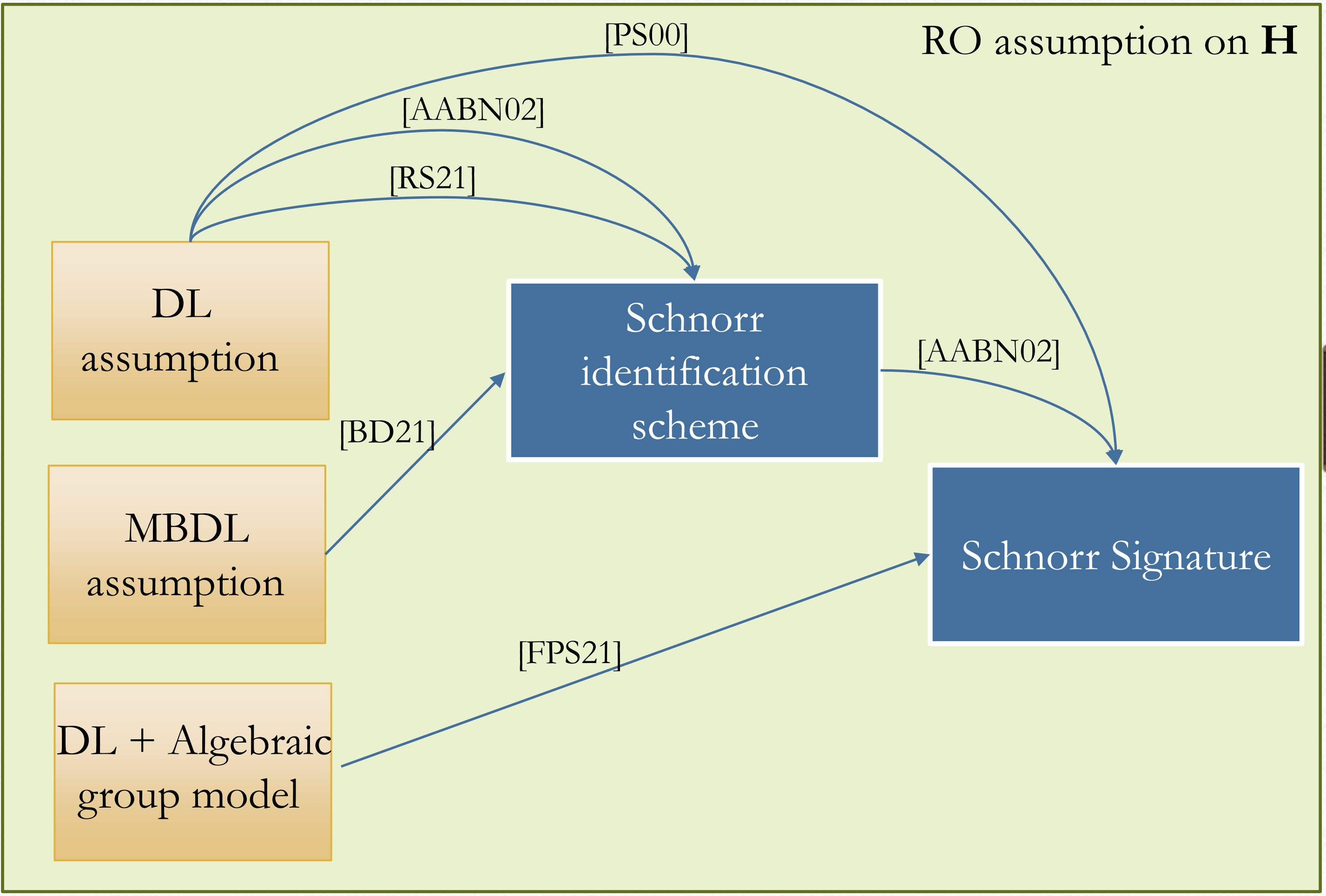
*curved arrows indicate non-tight reductions

Schnorr Signatures [Schnorr91]

Pros

- Simple
- Efficient (for a DL-based scheme)
- Short signatures
- Reducible to DL in the ROM
- Many formal security proofs with varying tightness & starting assumptions

Tighter reductions validate shorter parameters



RO assumption on **H**

DL assumption

MBDL assumption

DL + Algebraic group model

Schnorr identification scheme

Schnorr Signature

*curved arrows indicate non-tight reductions

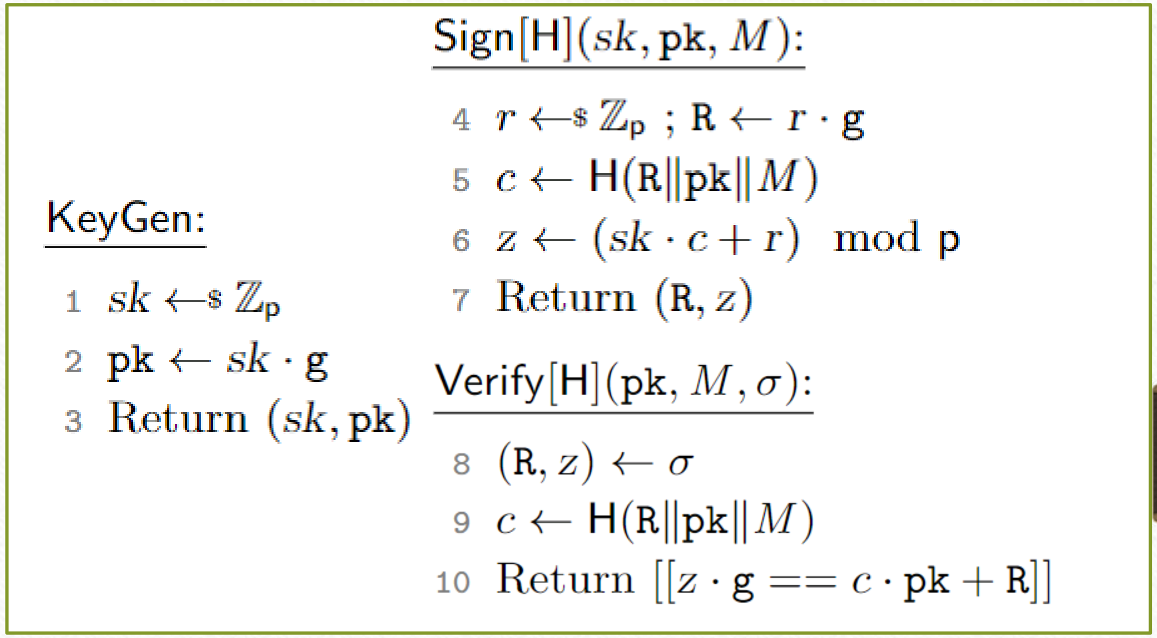
Schnorr Signatures [Schnorr91]

<ins>Pros</ins>

- Simple
- Efficient (for a DL-based scheme)
- Short signatures
- Reducible to DL in the ROM
- Many formal security proofs with varying tightness & starting assumptions

<ins>Cons</ins>

- Susceptible to randomness-reuse attack



```
KeyGen:
  $sk \leftarrow_\$ \mathbb{Z}_p$
  $pk \leftarrow sk \cdot g$
  Return $(sk, pk)$

Sign[H](sk, pk, M):
  $r \leftarrow_\$ \mathbb{Z}_p$ ; $R \leftarrow r \cdot g$
  $c \leftarrow H(R \| pk \| M)$
  $z \leftarrow (sk \cdot c + r) \mod p$
  Return $(R, z)$

Verify[H](pk, M, σ):
  $(R, z) \leftarrow \sigma$
  $c \leftarrow H(R \| pk \| M)$
  Return $[[z \cdot g == c \cdot pk + R]]$
```

Given signatures (R, z) and (R, z') on two different messages

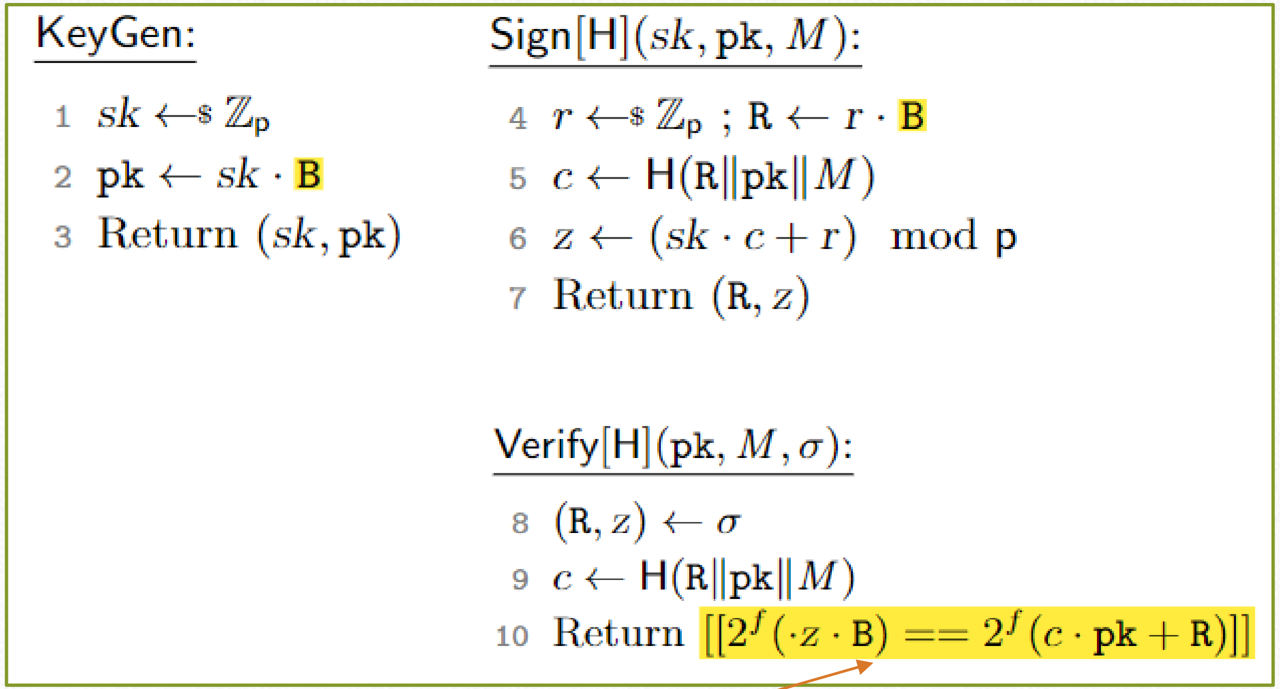
$$R = z \cdot g = (sk * c + r) \cdot g = z' \cdot g = (sk * c' + r) \cdot g$$

$$sk = \frac{z - z'}{c - c'}$$

EdDSA Signatures[BDLSY15]

EdDSA tweaks Schnorr for improved efficiency and security

- Choice of group:
    - Twisted Edwards curve
    - order $2^f \cdot \mathsf{p}$



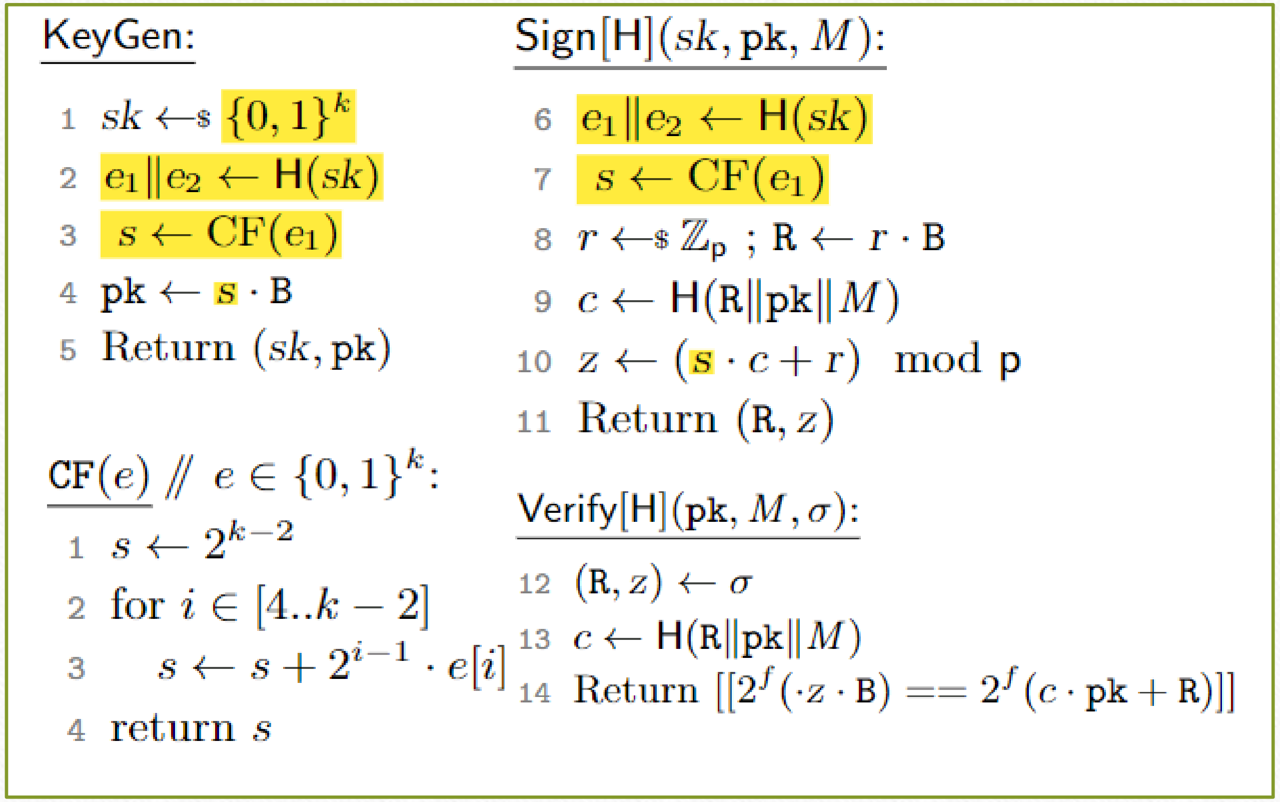
```
KeyGen:
    $sk \leftarrow_{\$} \mathbb{Z}_{\mathsf{p}}$
    $\mathsf{pk} \leftarrow sk \cdot \mathsf{B}$
    Return $(sk, \mathsf{pk})$

Sign[H]$(sk, \mathsf{pk}, M)$:
    $r \leftarrow_{\$} \mathbb{Z}_{\mathsf{p}}$ ; $\mathsf{R} \leftarrow r \cdot \mathsf{B}$
    $c \leftarrow \mathsf{H}(\mathsf{R} \| \mathsf{pk} \| M)$
    $z \leftarrow (sk \cdot c + r) \mod \mathsf{p}$
    Return $(\mathsf{R}, z)$

Verify[H]$(\mathsf{pk}, M, \sigma)$:
    $(\mathsf{R}, z) \leftarrow \sigma$
    $c \leftarrow \mathsf{H}(\mathsf{R} \| \mathsf{pk} \| M)$
    Return $[[2^f(\cdot z \cdot \mathsf{B}) == 2^f(c \cdot \mathsf{pk} + \mathsf{R})]]$
```

"permissive" verification vs
"strict" verification

EdDSA Signatures[BDLSY15]

EdDSA tweaks Schnorr for improved efficiency and security

- Choice of group:
  - Twisted Edwards curve
  - order $2^f \cdot \mathsf{p}$
- Hash RNG input and "clamp" secret keys

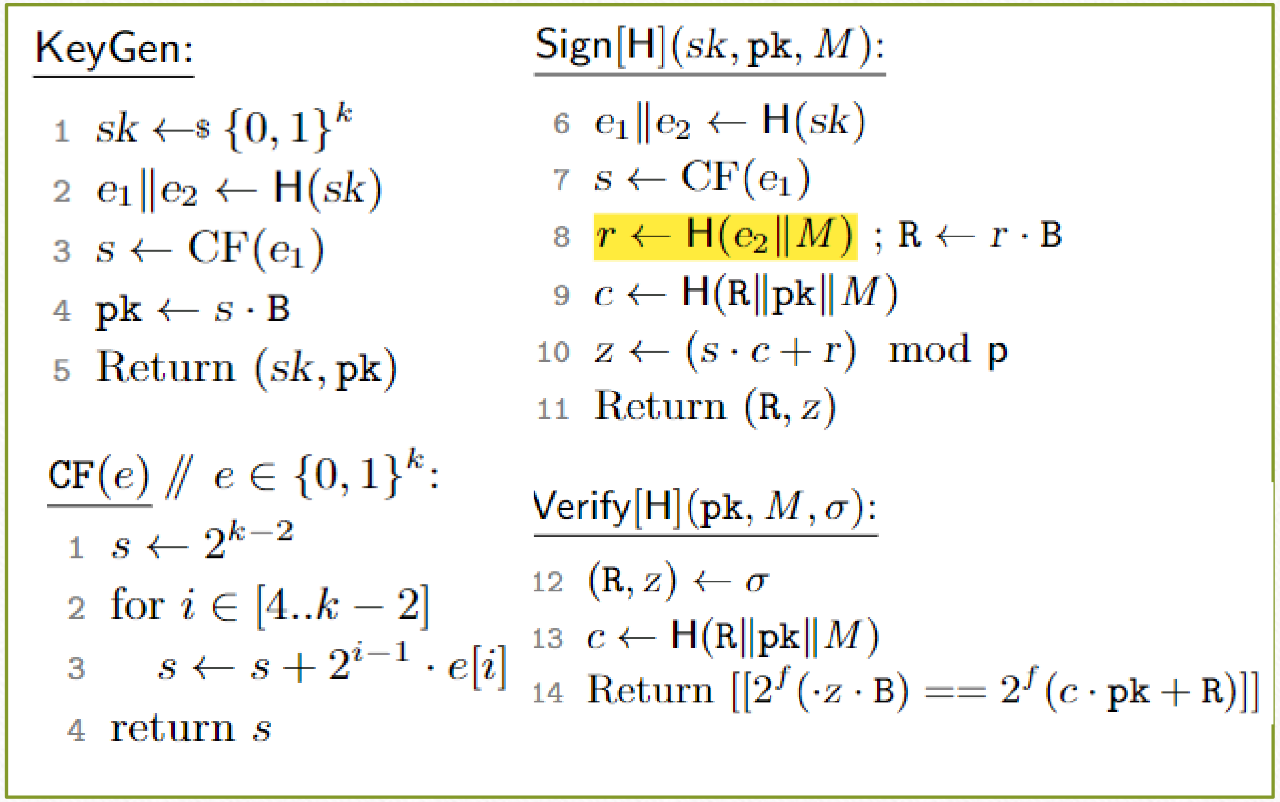


```
KeyGen:
  $sk \leftarrow_{\$} \{0,1\}^k$
  $e_1 \| e_2 \leftarrow \mathsf{H}(sk)$
  $s \leftarrow \mathrm{CF}(e_1)$
  $\mathsf{pk} \leftarrow s \cdot \mathsf{B}$
  Return $(sk, \mathsf{pk})$

$\mathrm{CF}(e)$ // $e \in \{0,1\}^k$:
  $s \leftarrow 2^{k-2}$
  for $i \in [4..k-2]$
      $s \leftarrow s + 2^{i-1} \cdot e[i]$
  return $s$

Sign[H]$(sk, \mathsf{pk}, M)$:
  $e_1 \| e_2 \leftarrow \mathsf{H}(sk)$
  $s \leftarrow \mathrm{CF}(e_1)$
  $r \leftarrow_{\$} \mathbb{Z}_{\mathsf{p}}$ ; $\mathsf{R} \leftarrow r \cdot \mathsf{B}$
  $c \leftarrow \mathsf{H}(\mathsf{R} \| \mathsf{pk} \| M)$
  $z \leftarrow (s \cdot c + r) \mod \mathsf{p}$
  Return $(\mathsf{R}, z)$

Verify[H]$(\mathsf{pk}, M, \sigma)$:
  $(\mathsf{R}, z) \leftarrow \sigma$
  $c \leftarrow \mathsf{H}(\mathsf{R} \| \mathsf{pk} \| M)$
  Return $[[2^f(\cdot z \cdot \mathsf{B}) == 2^f(c \cdot \mathsf{pk} + \mathsf{R})]]$
```

EdDSA Signatures[BDLSY15]

EdDSA tweaks Schnorr for improved efficiency and security

- Choice of group:
    - Twisted Edwards curve
    - order $2^f \cdot \mathsf{p}$
- Hash RNG input and “clamp” secret keys
- Derandomize Sign algorithm
  [Bar97][Wig97][NML97][Goldreich86][BPS16][BT16]



```
KeyGen:
  $sk \leftarrow_{\$} \{0,1\}^k$
  $e_1 \| e_2 \leftarrow \mathsf{H}(sk)$
  $s \leftarrow \mathrm{CF}(e_1)$
  $\mathsf{pk} \leftarrow s \cdot \mathsf{B}$
  Return $(sk, \mathsf{pk})$

CF$(e)$ // $e \in \{0,1\}^k$:
  $s \leftarrow 2^{k-2}$
  for $i \in [4..k-2]$
      $s \leftarrow s + 2^{i-1} \cdot e[i]$
  return $s$

Sign[H]$(sk, \mathsf{pk}, M)$:
  $e_1 \| e_2 \leftarrow \mathsf{H}(sk)$
  $s \leftarrow \mathrm{CF}(e_1)$
  $r \leftarrow \mathsf{H}(e_2 \| M)$ ; $\mathsf{R} \leftarrow r \cdot \mathsf{B}$
  $c \leftarrow \mathsf{H}(\mathsf{R} \| \mathsf{pk} \| M)$
  $z \leftarrow (s \cdot c + r) \mod \mathsf{p}$
  Return $(\mathsf{R}, z)$

Verify[H]$(\mathsf{pk}, M, \sigma)$:
  $(\mathsf{R}, z) \leftarrow \sigma$
  $c \leftarrow \mathsf{H}(\mathsf{R} \| \mathsf{pk} \| M)$
  Return $[[2^f(\cdot z \cdot \mathsf{B}) == 2^f(c \cdot \mathsf{pk} + \mathsf{R})]]$
```

EdDSA Signatures[BDLSY15]

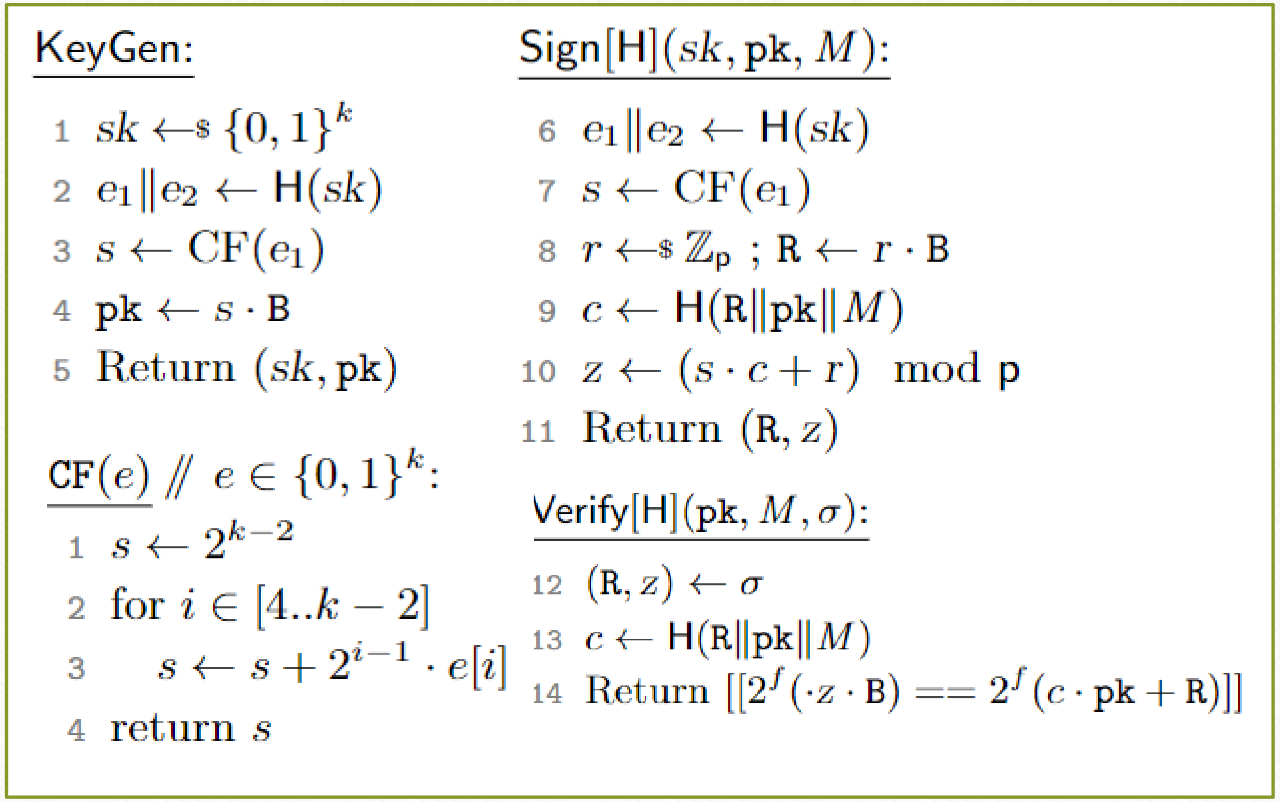
EdDSA tweaks Schnorr for improved efficiency and security

- Choice of group:
  - Twisted Edwards curve
  - order 2^b * p
- Hash RNG input and "clamp" secret keys
- Derandomize Sign algorithm
  [Bar97][Wig97][NML97][Goldreich86][BPS16][BT16]

EdDSA also specifies concrete choices of H

Ed25519	Ed448
SHA512	SHAKE

Can these be modeled as random oracles?



```
KeyGen:
    $sk \leftarrow_{\$} \{0,1\}^k$
    $e_1 \| e_2 \leftarrow \mathsf{H}(sk)$
    $s \leftarrow \mathrm{CF}(e_1)$
    $\mathsf{pk} \leftarrow s \cdot \mathsf{B}$
    Return $(sk, \mathsf{pk})$

CF(e) // $e \in \{0,1\}^k$:
    $s \leftarrow 2^{k-2}$
    for $i \in [4..k-2]$
        $s \leftarrow s + 2^{i-1} \cdot e[i]$
    return $s$

Sign[H]($sk$, pk, $M$):
    $e_1 \| e_2 \leftarrow \mathsf{H}(sk)$
    $s \leftarrow \mathrm{CF}(e_1)$
    $r \leftarrow_{\$} \mathbb{Z}_p$ ; $\mathsf{R} \leftarrow r \cdot \mathsf{B}$
    $c \leftarrow \mathsf{H}(\mathsf{R} \| \mathsf{pk} \| M)$
    $z \leftarrow (s \cdot c + r) \mod \mathsf{p}$
    Return $(\mathsf{R}, z)$

Verify[H](pk, $M$, $\sigma$):
    $(\mathsf{R}, z) \leftarrow \sigma$
    $c \leftarrow \mathsf{H}(\mathsf{R} \| \mathsf{pk} \| M)$
    Return $[[2^f(\cdot z \cdot \mathsf{B}) == 2^f(c \cdot \mathsf{pk} + \mathsf{R})]]$
```

Instantiating a Random Oracle

SHA512

Compression function sha512

SHAKE

Permutation

Do these functions behave like random oracles? **No**

Instantiating a Random Oracle

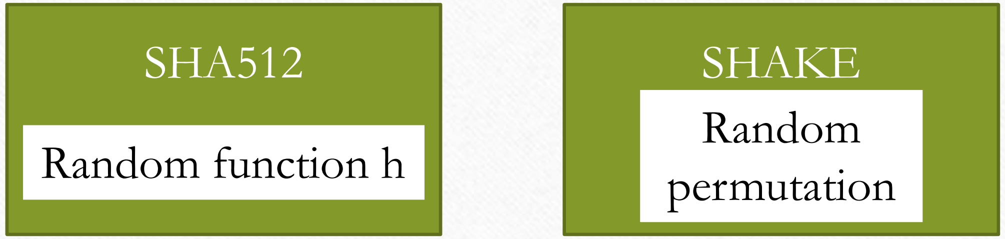
SHA512

Compression function sha512

SHAKE

Permutation

Do these functions behave like random oracles? **No**



SHA512

Random function h

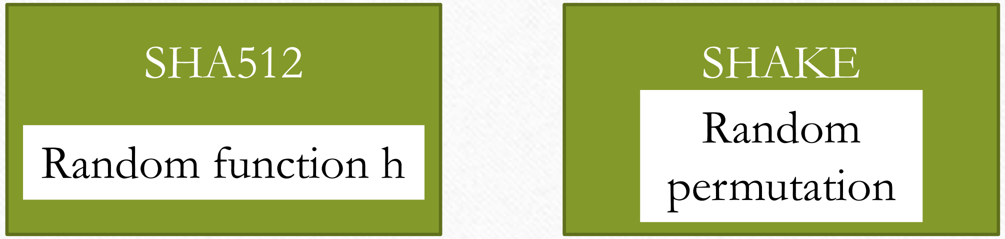
SHAKE

Random permutation

Do these functions behave like random oracles?

Instantiating a Random Oracle

Do these functions behave like random oracles? **No**

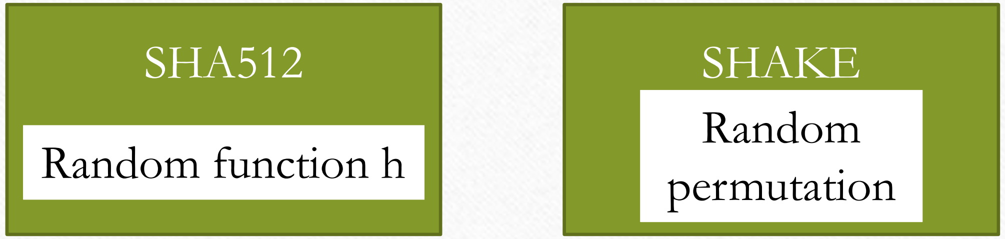


Do these functions behave like random oracles? ←

Are SHA512 and SHAKE **indifferentiable** from a random oracle?[MRH04]

Instantiating a Random Oracle

Do these functions behave like random oracles? **No**



No[CDMP05] **Yes[BDVA08]**

Do **these** functions behave like random oracles?

Instantiating a Random Oracle

SHA512

Compression function sha512

SHAKE

Permutation

Do these functions behave like random oracles? **No**

SHA512

Random function h

No[CDMP05]

SHAKE

Random permutation

Yes[BDVA08]

Do **these** functions behave like random oracles?

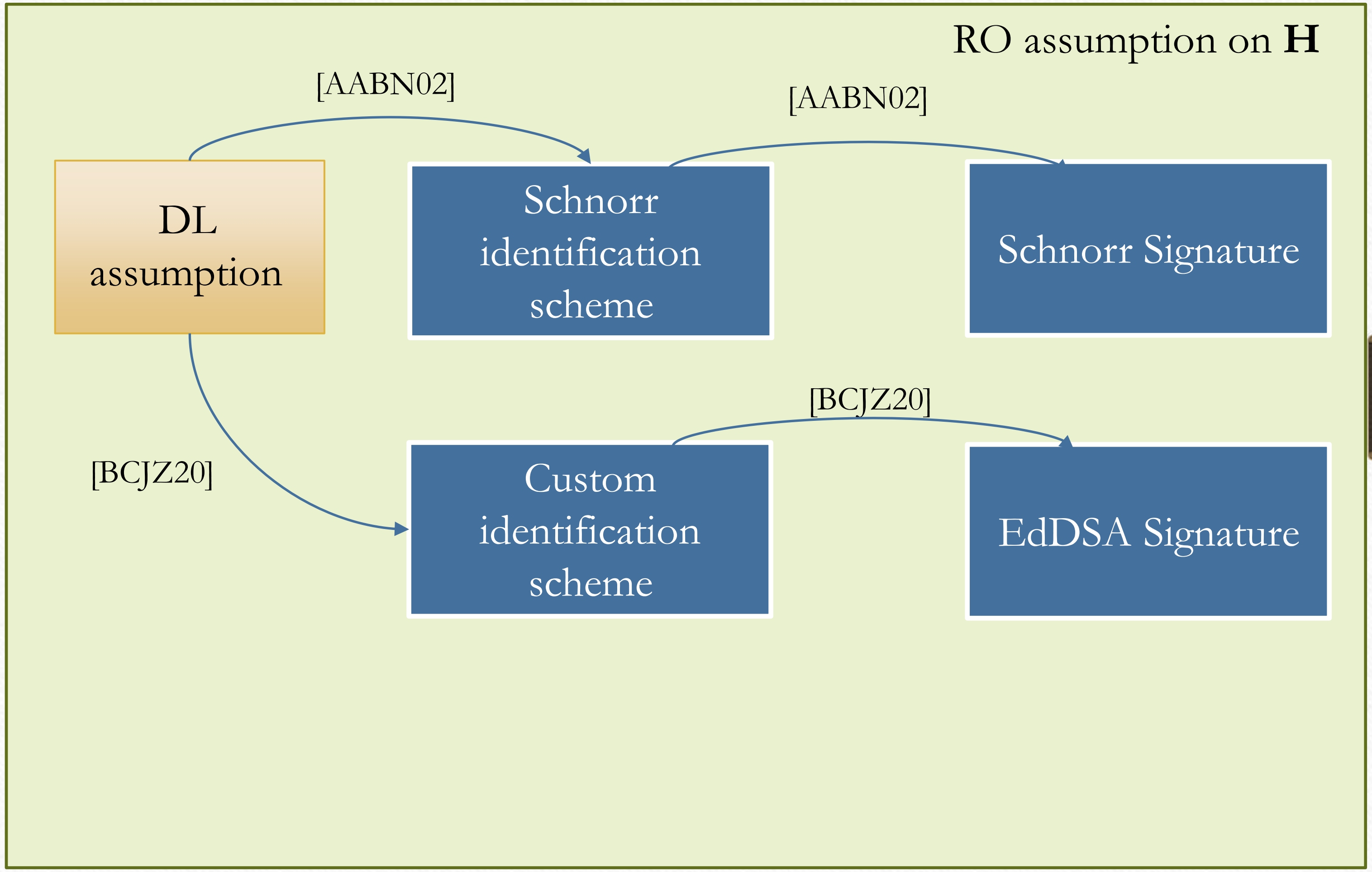
Length Extension Attack on SHA512

Given messages m1 and m2 and compression function h

$$\mathrm{SHA512}(m_1 || m_2) = \mathrm{sha512}(\mathrm{SHA512}(m_1) || m_2)$$

Does this make Ed25519 insecure? **No**.
But it does mean that SHA512 should not be modeled as a random oracle.

Security Analysis of EdDSA



RO assumption on **H**

DL assumption

[AABN02]

Schnorr identification scheme

[AABN02]

Schnorr Signature

[BCJZ20]

Custom identification scheme

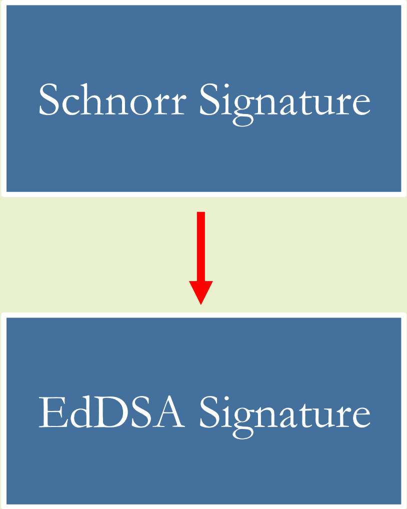
[BCJZ20]

EdDSA Signature

Our Contributions

A new proof of security for EdDSA

- **Reduce directly to security of Schnorr signatures**
  - Simpler, more modular analysis
  - Can leverage recent tighter bounds for Schnorr



RO assumption on **H**

Schnorr Signature

EdDSA Signature

Our Contributions

A new proof of security for EdDSA

- **Reduce directly to security of Schnorr signatures**
  - Simpler, more modular analysis
  - Can leverage recent tighter bounds for Schnorr

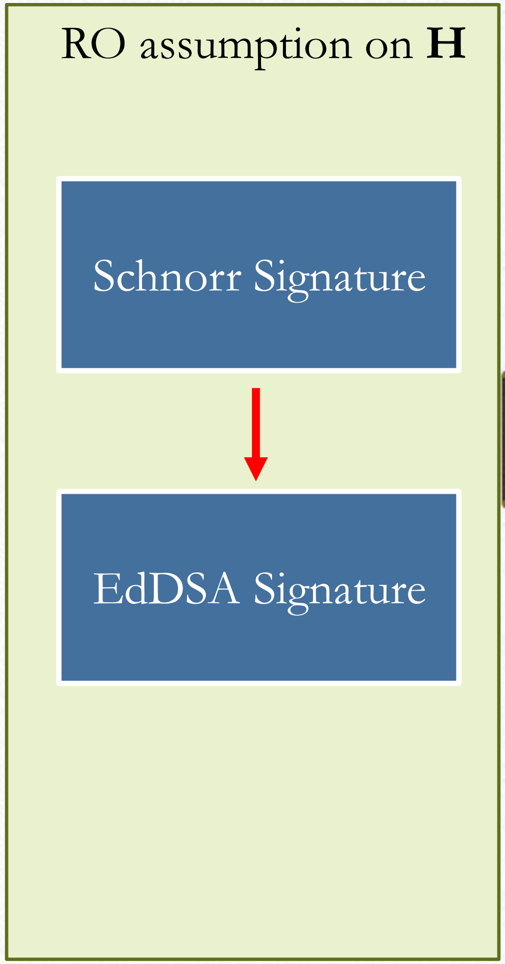
Ex: If attacker **A** performs up to 2^{70} operations and 2^{60} oracle queries, and curve **x25519** has order $\approx 2^{252}$

Its DL advantage is at most $2^{\wedge -112}$ [Shoup97]
Its UF-CMA advantage against Schnorr is at most

- 2^{-41} assuming DL [RS21]
- 2^{-52} assuming MBDL[BD21]
- 2^{-130} assuming DL in the AGM[FPS19]

Its UF-CMA advantage against Ed25519 is at most

- 2^{-26} by [BCJZ20]
- **2^{-37} by [BDD23]**
- **2^{-48} by [BDD23] assuming MBDL**
- **2^{-126} by [BDD23] assuming DL + AGM**


RO assumption on **H**

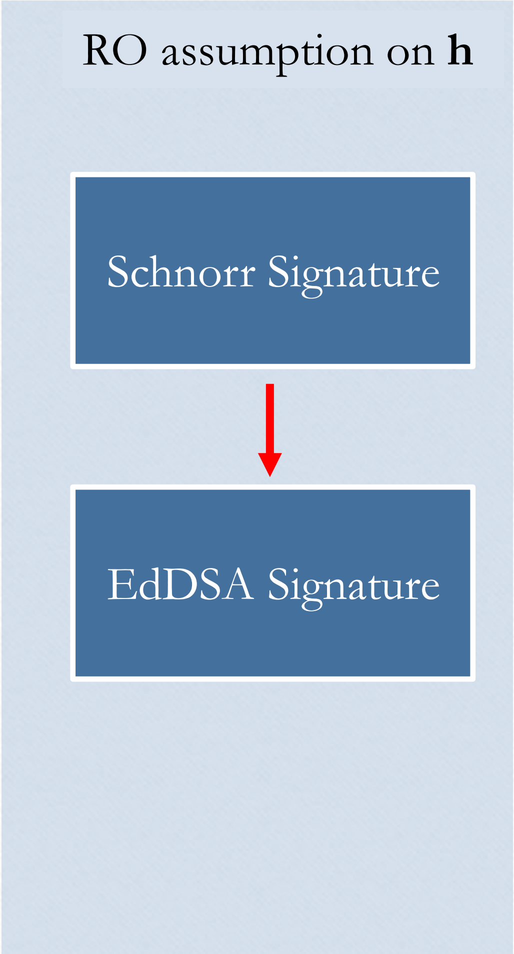
Schnorr Signature

EdDSA Signature

Our Contributions

A new proof of security for EdDSA

- **Reduce directly to security of Schnorr signatures**
    - Simpler, more modular analysis
    - Can leverage recent tighter bounds for Schnorr
- **Weaker ROM assumption**
    - Idealize only compression function/permutation
    - Rely on standard-model properties where possible
    - Bounds attackers who use extension attack


RO assumption on **h**

Schnorr Signature

EdDSA Signature

Our contributions

A new proof of security for EdDSA

- **Reduce directly to security of Schnorr signatures**
    - Simpler, more modular analysis
    - Can leverage recent tighter bounds for Schnorr
- **Weaker ROM assumption**
    - Idealize only compression function/permutation
    - Rely on standard-model properties where possible
    - Bounds attackers who use extension attack

+ some handy generic results

Derive-then-Derandomize Transform:
A generic signature-hardening transform that captures EdDSA's tweaks

Our contributions

A new proof of security for EdDSA

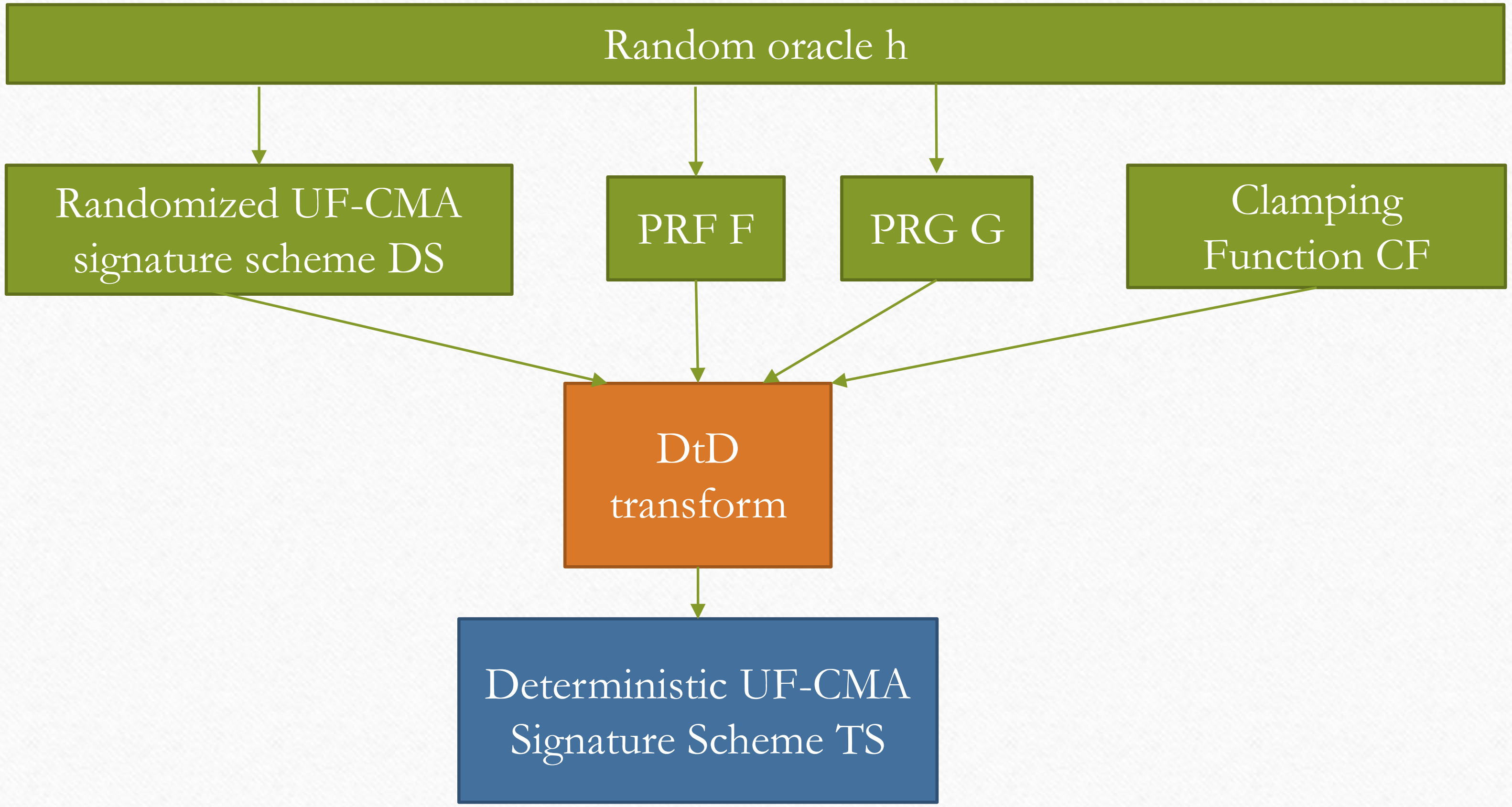
- **Reduce directly to security of Schnorr signatures**
  - Simpler, more modular analysis
  - Can leverage recent tighter bounds for Schnorr
- **Weaker ROM assumption**
  - Idealize only compression function/permutation
  - Rely on standard-model properties where possible
  - Bounds attackers who use extension attack

+ some handy generic results

Derive-then-Derandomize Transform: A generic signature-hardening transform that captures EdDSA's tweaks

Improved indifferentiability analysis for the **Shrink-MD hash function class** that transforms the output of an MD hash, **including chop-MD**

Derive-then-Derandomize transform



Random oracle h

Randomized UF-CMA signature scheme DS

PRF F

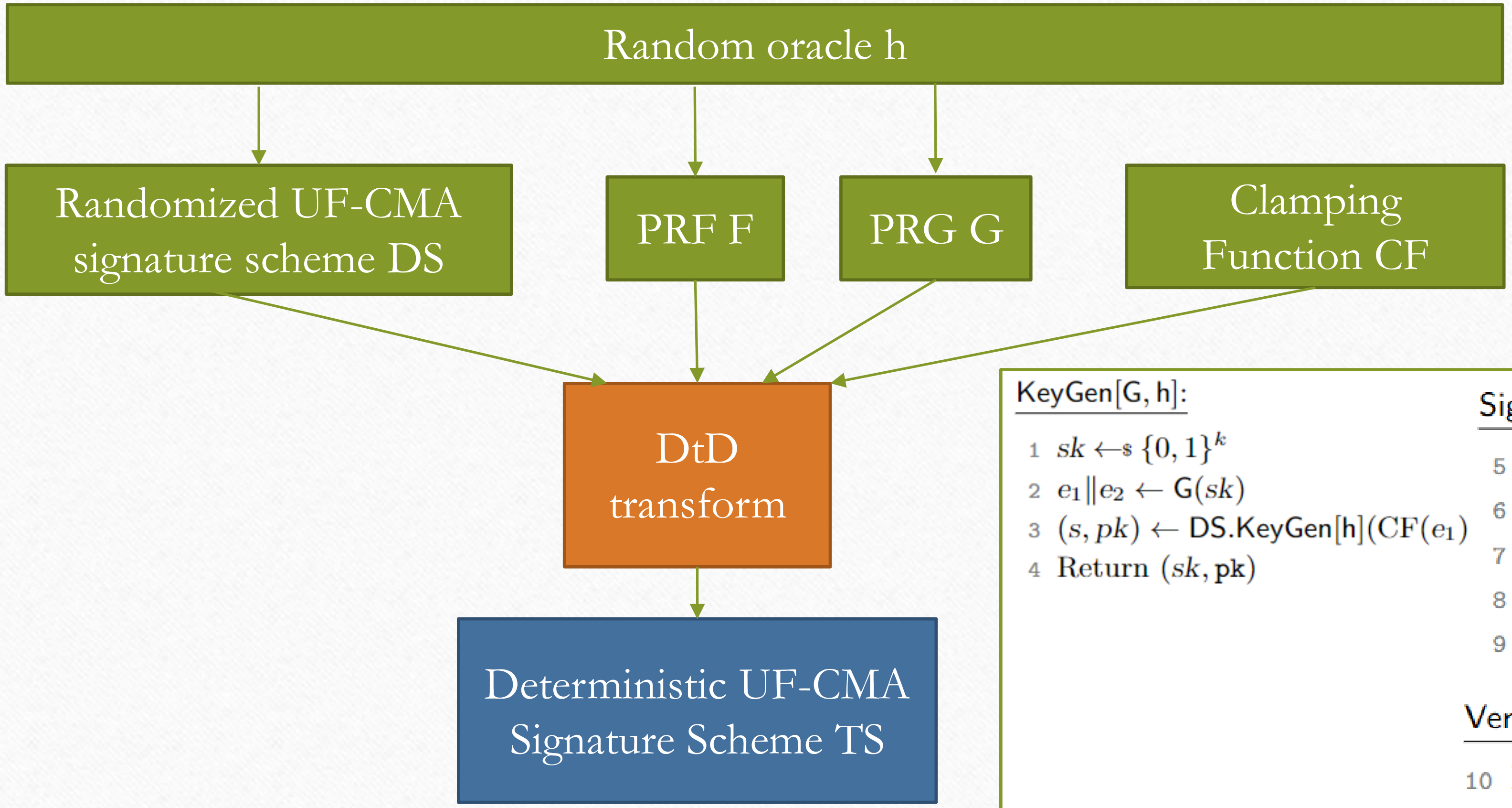
PRG G

Clamping Function CF

DtD transform

Deterministic UF-CMA Signature Scheme TS

Derive-then-Derandomize transform



Random oracle h

Randomized UF-CMA signature scheme DS

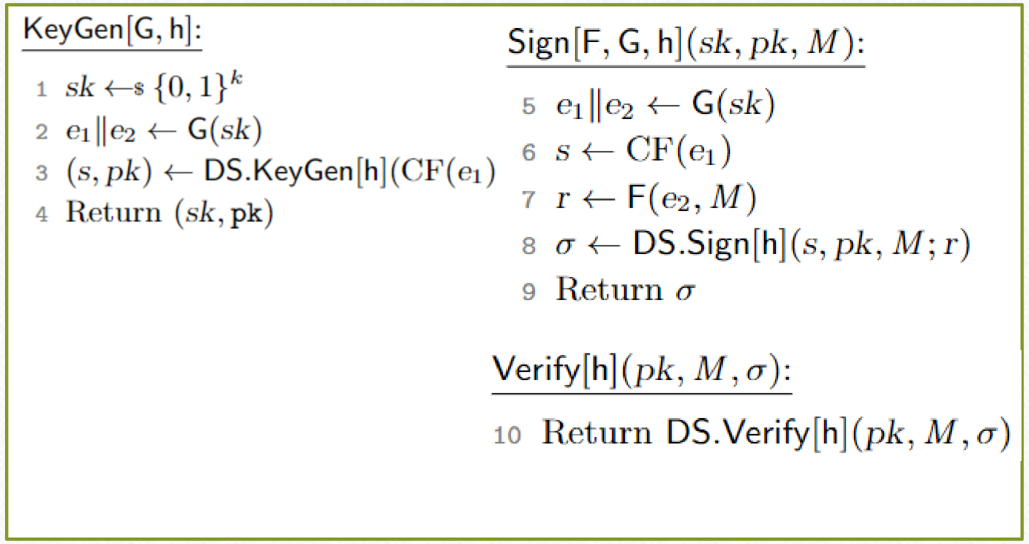
PRF F

PRG G

Clamping Function CF

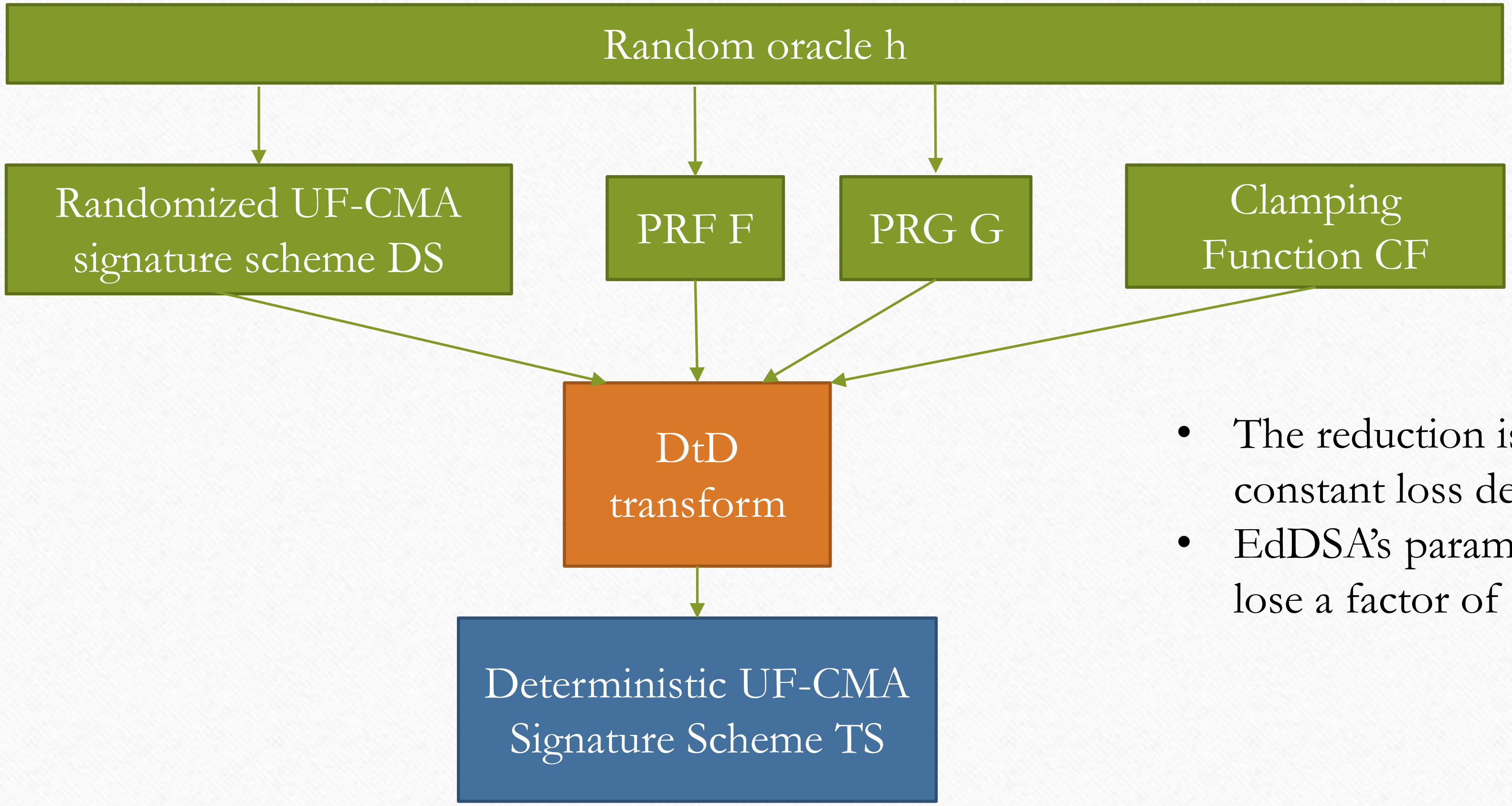
DtD transform

Deterministic UF-CMA Signature Scheme TS



```
$\mathsf{KeyGen}[\mathsf{G},\mathsf{h}]$:
    $sk \leftarrow\!\!\$\ \{0,1\}^k$
    $e_1 \| e_2 \leftarrow \mathsf{G}(sk)$
    $(s, pk) \leftarrow \mathsf{DS.KeyGen}[\mathsf{h}](\mathrm{CF}(e_1))$
    Return $(sk, \mathsf{pk})$

$\mathsf{Sign}[\mathsf{F},\mathsf{G},\mathsf{h}](sk, pk, M)$:
    $e_1 \| e_2 \leftarrow \mathsf{G}(sk)$
    $s \leftarrow \mathrm{CF}(e_1)$
    $r \leftarrow \mathsf{F}(e_2, M)$
    $\sigma \leftarrow \mathsf{DS.Sign}[\mathsf{h}](s, pk, M; r)$
    Return $\sigma$

$\mathsf{Verify}[\mathsf{h}](pk, M, \sigma)$:
    Return $\mathsf{DS.Verify}[\mathsf{h}](pk, M, \sigma)$
```

Derive-then-Derandomize transform

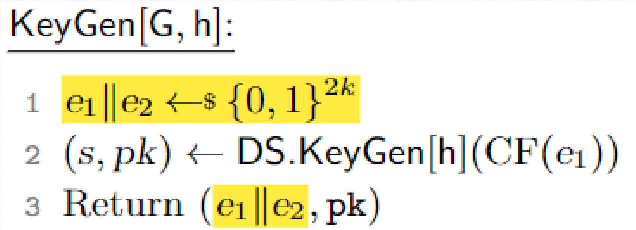


- The reduction is tight, with only constant loss depending on CF
- EdDSA’s parameters cause it to lose a factor of 16

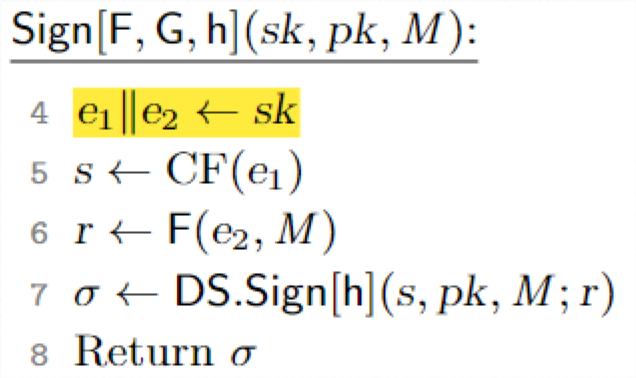
Proving security for DtD

We reverse the transform step-by-step

1



```
KeyGen[G, h]:
    $e_1 \| e_2 \leftarrow\$ \{0,1\}^{2k}$
    $(s, pk) \leftarrow \mathsf{DS.KeyGen}[\mathsf{h}](\mathrm{CF}(e_1))$
    Return $(e_1 \| e_2, \mathsf{pk})$
```

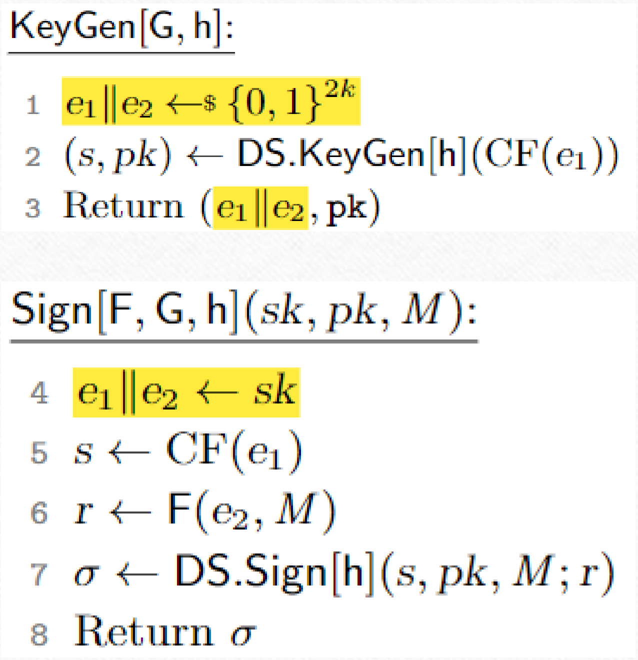


```
Sign[F, G, h](sk, pk, M):
    $e_1 \| e_2 \leftarrow sk$
    $s \leftarrow \mathrm{CF}(e_1)$
    $r \leftarrow \mathsf{F}(e_2, M)$
    $\sigma \leftarrow \mathsf{DS.Sign}[\mathsf{h}](s, pk, M; r)$
    Return $\sigma$
```

Proving security for DtD

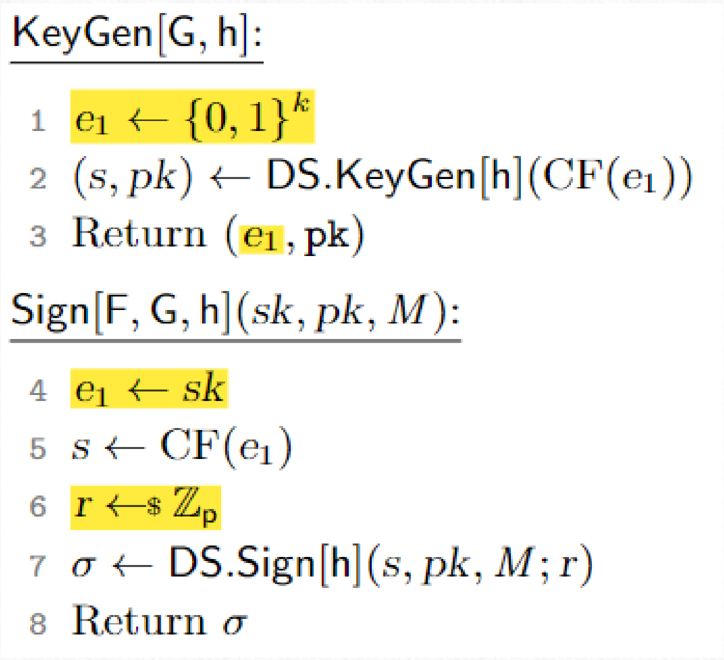
We reverse the transform step-by-step

1



```
KeyGen[G, h]:
    $e_1 \| e_2 \leftarrow\!\$\ \{0,1\}^{2k}$
    $(s, pk) \leftarrow$ DS.KeyGen[h](CF($e_1$))
    Return ($e_1 \| e_2$, pk)

Sign[F, G, h]($sk, pk, M$):
    $e_1 \| e_2 \leftarrow sk$
    $s \leftarrow$ CF($e_1$)
    $r \leftarrow$ F($e_2, M$)
    $\sigma \leftarrow$ DS.Sign[h]($s, pk, M; r$)
    Return $\sigma$
```

2

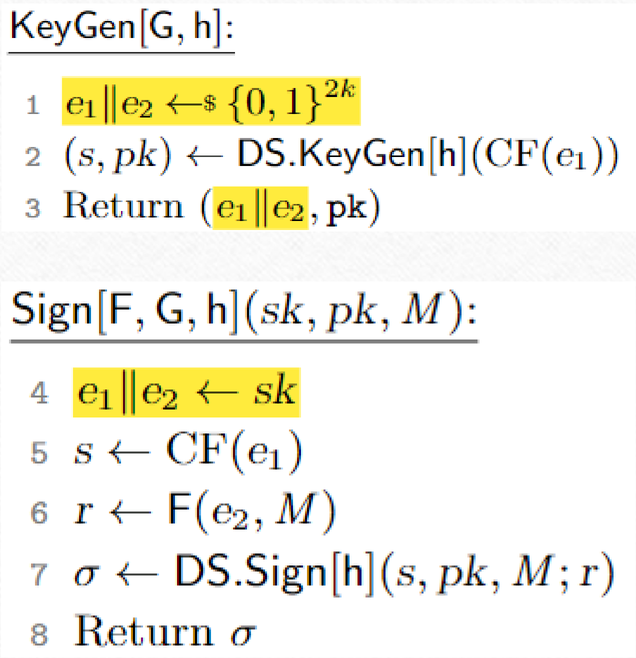


```
KeyGen[G, h]:
    $e_1 \leftarrow \{0,1\}^{k}$
    $(s, pk) \leftarrow$ DS.KeyGen[h](CF($e_1$))
    Return ($e_1$, pk)

Sign[F, G, h]($sk, pk, M$):
    $e_1 \leftarrow sk$
    $s \leftarrow$ CF($e_1$)
    $r \leftarrow\!\$\ \mathbb{Z}_p$
    $\sigma \leftarrow$ DS.Sign[h]($s, pk, M; r$)
    Return $\sigma$
```

Proving security for DtD

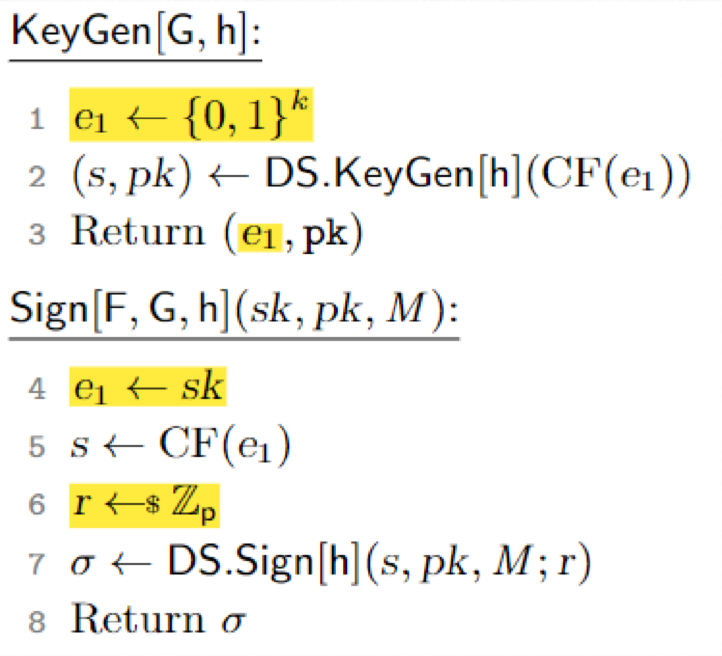
We reverse the transform step-by-step

①



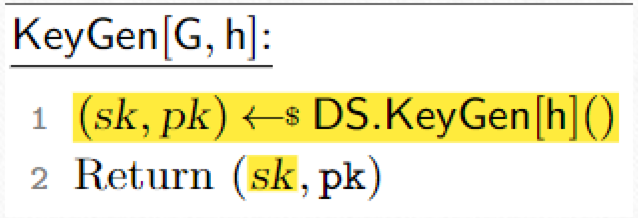
```
KeyGen[G, h]:
    $e_1 \| e_2 \leftarrow_{\$} \{0,1\}^{2k}$
    $(s, pk) \leftarrow \mathsf{DS.KeyGen}[\mathsf{h}](\mathrm{CF}(e_1))$
    Return $(e_1 \| e_2, \mathsf{pk})$

Sign[F, G, h](sk, pk, M):
    $e_1 \| e_2 \leftarrow sk$
    $s \leftarrow \mathrm{CF}(e_1)$
    $r \leftarrow \mathsf{F}(e_2, M)$
    $\sigma \leftarrow \mathsf{DS.Sign}[\mathsf{h}](s, pk, M; r)$
    Return $\sigma$
```

②



```
KeyGen[G, h]:
    $e_1 \leftarrow \{0,1\}^{k}$
    $(s, pk) \leftarrow \mathsf{DS.KeyGen}[\mathsf{h}](\mathrm{CF}(e_1))$
    Return $(e_1, \mathsf{pk})$

Sign[F, G, h](sk, pk, M):
    $e_1 \leftarrow sk$
    $s \leftarrow \mathrm{CF}(e_1)$
    $r \leftarrow_{\$} \mathbb{Z}_{\mathsf{p}}$
    $\sigma \leftarrow \mathsf{DS.Sign}[\mathsf{h}](s, pk, M; r)$
    Return $\sigma$
```

③



```
KeyGen[G, h]:
    $(sk, pk) \leftarrow_{\$} \mathsf{DS.KeyGen}[\mathsf{h}]()$
    Return $(sk, \mathsf{pk})$
```

Lose a factor of
$|\mathrm{Im}(\mathrm{CF})| / |\mathrm{K}|$

Instantiating F, G, and H

To cast EdDSA as the output of a DtD transform, we must define DS = Schnorr and

Function	Desired security	Instantiation in EdDSA
F	Pseudorandom function	**F(sk, M) = H(sk | | M) mod p**
G	Pseudorandom generator	**G(sk) = H(sk)**
H	Random oracle	**H(R| |A| |M) = H(R| |A| |M) mod p**

Can we achieve the desired security when **H** is an **MD** hash like **SHA512** if we assume the compression function is ideal?

Instantiating F, G, and H

To cast EdDSA as the output of a DtD transform, we must define DS = Schnorr and

Function	Desired security	Instantiation in EdDSA
F	Pseudorandom function	**F(sk, M) = H(sk || M) mod p**
G	Pseudorandom generator	**G(sk) = H(sk)**
H	Random oracle	**H(R||A||M) = H(R||A||M) mod p**

Yes, this is AMAC[BBT16]

Can we achieve the desired security when **H** is an **MD** hash like **SHA512** if we assume the compression function is ideal?

Instantiating F, G, and H

To cast EdDSA as the output of a DtD transform, we must define DS = Schnorr and

Function	Desired security	Instantiation in EdDSA
F	Pseudorandom function	**F(sk, M) = H(sk || M) mod p**
G	Pseudorandom generator	**G(sk) = H(sk)**
H	Random oracle	**H(R||A||M) = H(R||A||M) mod p**

Yes, this is AMAC[BBT16]

Yes, this is easily shown

Can we achieve the desired security when **H** is an **MD** hash like **SHA512** if we assume the compression function is ideal?

Instantiating F, G, and H

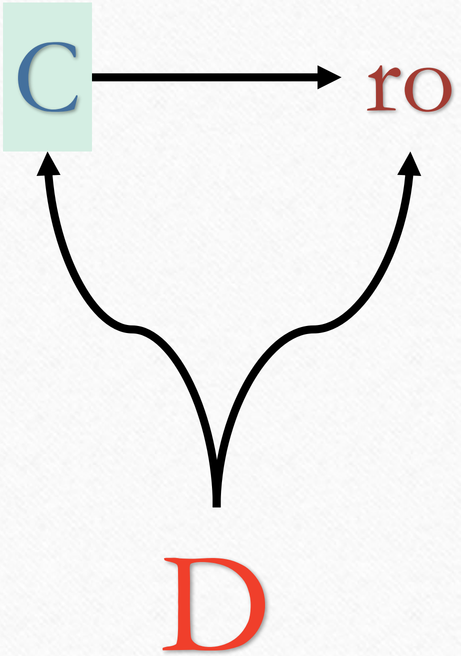
To cast EdDSA as the output of a DtD transform, we must define DS = Schnorr and

Function	Desired security	Instantiation in EdDSA	
F	Pseudorandom function	$\mathbf{F(sk, M) = H(sk \Vert M) \bmod p}$	Yes, this is AMAC[BBT16]
G	Pseudorandom generator	$\mathbf{G(sk) = H(sk)}$	Yes, this is easily shown
H	Random oracle	$\mathbf{H(R \Vert A \Vert M) = H(R \Vert A \Vert M) \bmod p}$	**Yes, we prove this**

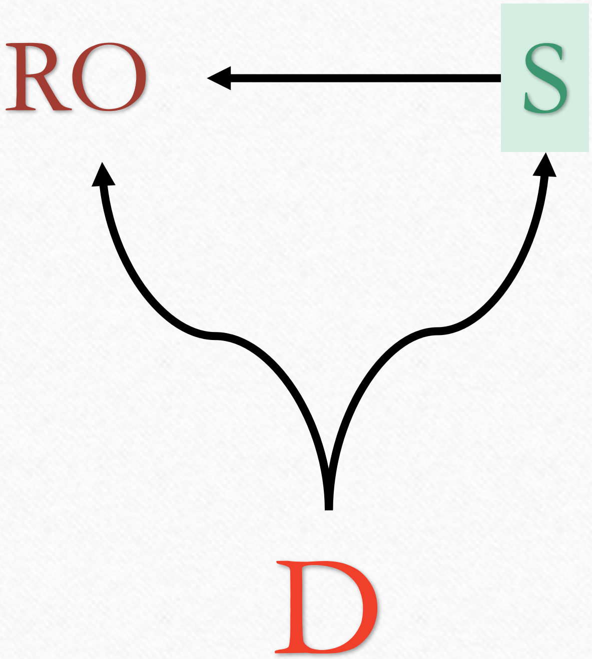
Can we achieve the desired security when **H** is an **MD** hash like **SHA512** if we assume the compression function is ideal?

Indifferentiability [MRH04]

Real World



Ideal World

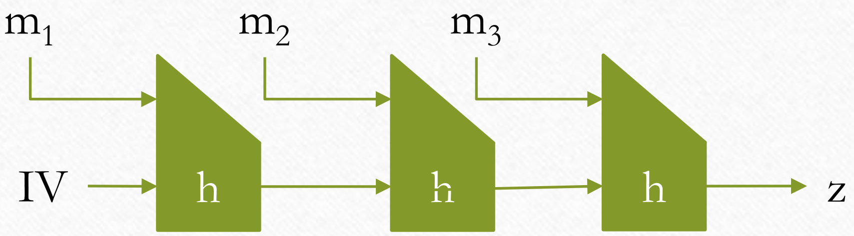


S is a simulator, an algorithm whose job is to imitate the random oracle H

We say **C** is indifferentiable with respect to a simulator **S** if distinguisher D cannot tell which world it is in.

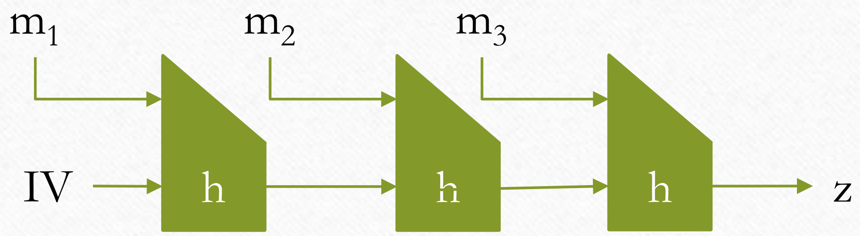
Merkle-Damgard Hash Structure

- SHA512 is a Merkle-Damgard hash function based on a compression function h



Merkle-Damgard Hash Structure

- SHA512 is a Merkle-Damgard hash function based on a compression function h



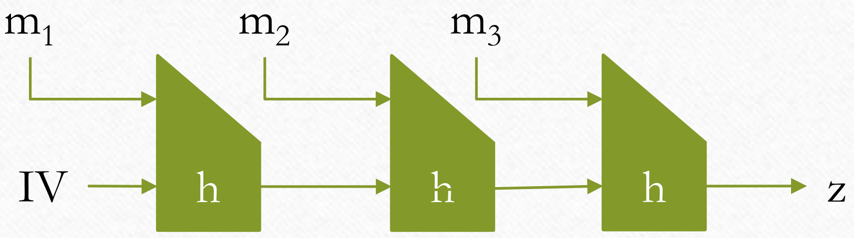
[CDMP05] MD hash is not indifferentiable, but **chop-MD** is.

$$\mathbf{Chop\text{-}MD[h](M) = MD[h](M) \bmod 2^c}$$

This is almost the result we need, but replacing 2^c with p introduces **bias**.

Merkle-Damgard Hash Structure

- SHA512 is a Merkle-Damgard hash function based on a compression function h



[CDMP05] MD hash is not indifferentiable, but **chop-MD** is.

Chop-MD[h](M) = MD[h](M) mod 2^c

This is almost the result we need, but replacing 2^c with p introduces **bias**.

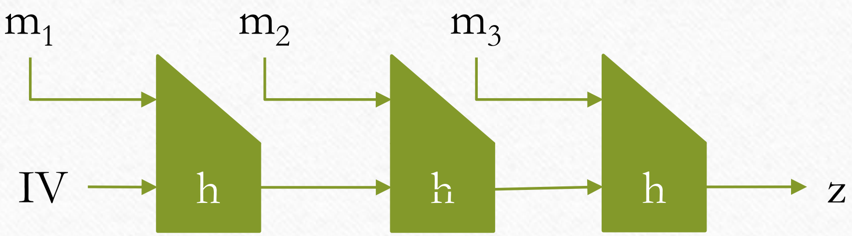
Shrink-MD[h, Out](M) = Out(MD[h](M))

3 conditions on **Out:**

- Reversibility: we can sample from the preimage set
- Quasi-regularity: Every point in the image set S has many preimages
- Near-Uniformity: $D := z \leftarrow_{\$} \mathsf{Out}^{-1}(y) : y \leftarrow_{\$} S$ is close to the uniform distribution

Merkle-Damgard Hash Structure

- SHA512 is a Merkle-Damgard hash function based on a compression function h



[CDMP05] MD hash is not indifferentiable, but **chop-MD** is.

Chop-MD[h](M) = MD[h](M) mod 2^c

This is almost the result we need, but replacing 2^c with p introduces **bias**.

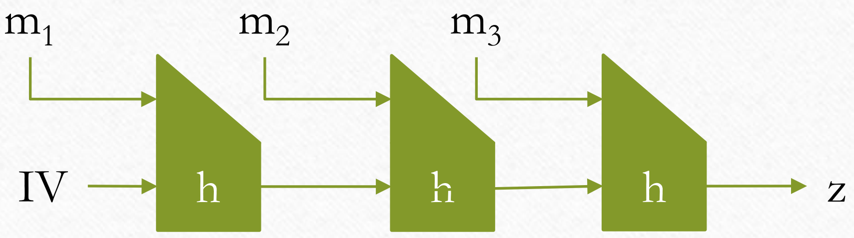
Shrink-MD[h, Out](M) = Out(MD[h](M))

<u>3 conditions on **Out:**</u>

- Reversibility: we can sample from the preimage set
- Quasi-regularity: Every point in the image set S has many preimages
- Near-Uniformity: $D := z \leftarrow_{\$} \mathsf{Out}^{-1}(y) : y \leftarrow_{\$} S$ is close to the uniform distribution

Merkle-Damgard Hash Structure

- SHA512 is a Merkle-Damgard hash function based on a compression function h



[CDMP05] MD hash is not indifferentiable, but **chop-MD** is.

Chop-MD[h](M) = MD[h](M) mod 2^c

This is almost the result we need, but replacing 2^c with p introduces **bias**.

Shrink-MD[h, Out](M) = Out(MD[h](M))

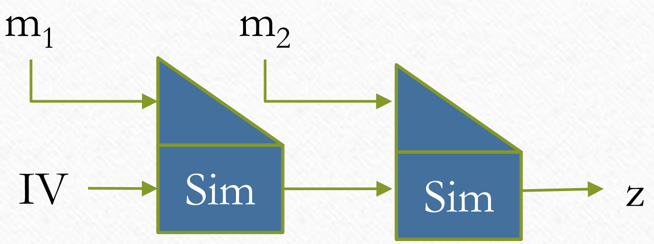
3 conditions on **Out:**

- Reversibility: we can sample from the preimage set
- Quasi-regularity: Every point in the image set S has many preimages
- Near-Uniformity:
  $D := z \leftarrow_{\$} \mathsf{Out}^{-1}(y) : y \leftarrow_{\$} S$ is close to the uniform distribution

We prove **indifferentiability** for any **Shrink-MD** construction, including **Chop-MD** and MD mod p

The Indifferentiability of Shrink-MD

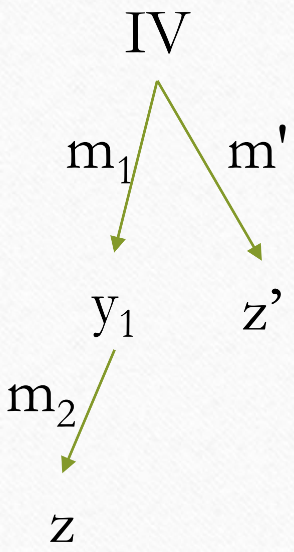
To show that a Shrink-MD hash function is indifferentiable,
we must **consistently simulate a random compression function**



Prior simulators for **chop-MD** construct a **tree** to store all the queries.

The proofs **add extra nodes** to this tree that are **detectable** in certain situations

We solve this problem by constructing **two trees** in our simulator:
one to answer adversarial queries, and one to track the extra nodes



Our contributions

+ some handy generic results

A new proof of security for EdDSA

- **Reduce directly to security of Schnorr signatures**
  - Simpler, more modular analysis
  - Can leverage recent tighter bounds for Schnorr
- **Weaken ROM assumption**
  - Use indifferentiability to idealize only compression function/permutation
  - Rely on standard-model properties where possible
  - Explicitly capture length-extension attack

Derive-then-Derandomize Transform: A generic signature-hardening transform that captures EdDSA's tweaks

Improved indifferentiability analysis for the **Shrink-MD hash function class** that transforms the output of an MD hash, **including chop-MD**

Thank you!