

Simple Extremely Lossy Functions from Small-Exponent Hashing



Damiano Abram



THE UNIVERSITY
of EDINBURGH



Agni Datta



THE UNIVERSITY
of EDINBURGH



Archisman Dutta



AARHUS UNIVERSITY



Lawrence Roy



AU CRYPTO SEMINAR • 15 October 2026

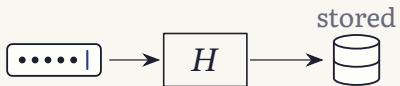
ePrint 2026/2367

MOTIVATION



Why would anyone want a hash function that throws information away?

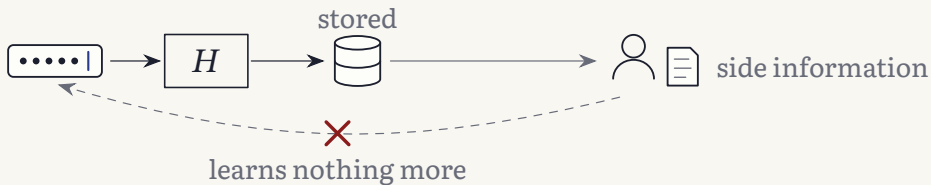
Password hashing



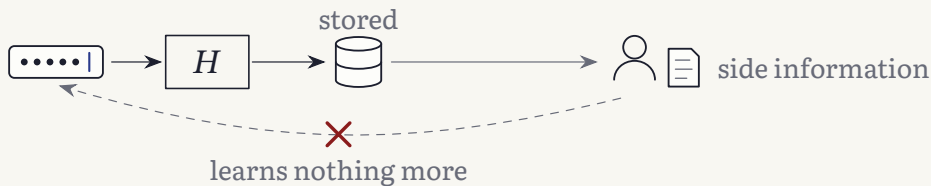
Password hashing



Password hashing



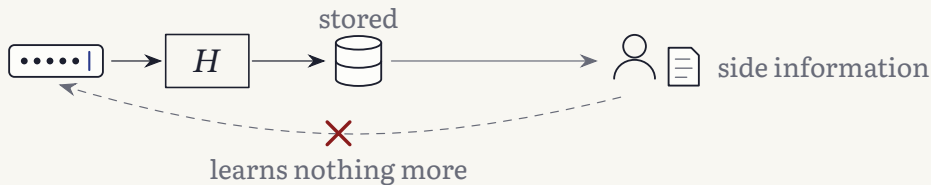
Password hashing



WITH A RANDOM ORACLE



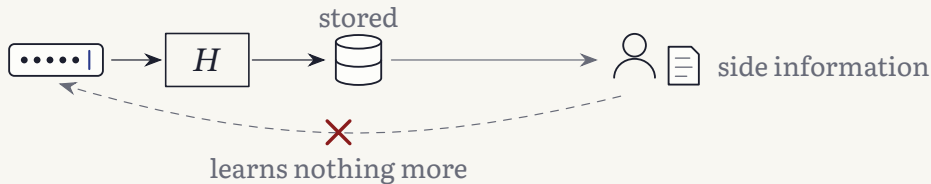
Password hashing



WITH A RANDOM ORACLE



Password hashing



WITH A RANDOM ORACLE



Formally: auxiliary-input point obfuscation. ELFs give it [Zha16].

Random oracles and real hash functions

random oracle

$$x_0 \mapsto H(x_0)$$

$$x_1 \mapsto H(x_1)$$

$$x_2 \mapsto H(x_2)$$

$$x_3 \mapsto H(x_3)$$

$$x_4 \mapsto H(x_4)$$

$$x_5 \mapsto H(x_5)$$

$$x_6 \mapsto H(x_6)$$

Random oracles and real hash functions

random oracle

$x_0 \mapsto H(x_0)$
$x_1 \mapsto H(x_1)$
$x_2 \mapsto H(x_2)$
$x_3 \mapsto H(x_3)$
$x_4 \mapsto H(x_4)$
$x_5 \mapsto H(x_5)$
$x_6 \mapsto H(x_6)$

the adversary sees a few rows

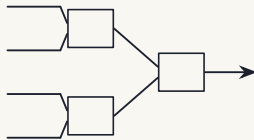
Random oracles and real hash functions

random oracle

$x_0 \mapsto H(x_0)$
$x_1 \mapsto H(x_1)$
$x_2 \mapsto H(x_2)$
$x_3 \mapsto H(x_3)$
$x_4 \mapsto H(x_4)$
$x_5 \mapsto H(x_5)$
$x_6 \mapsto H(x_6)$

the adversary sees a few rows

real hash function



a small circuit

Random oracles and real hash functions

random oracle

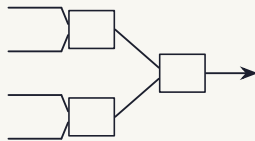
$x_0 \mapsto H(x_0)$
$x_1 \mapsto H(x_1)$
$x_2 \mapsto H(x_2)$
$x_3 \mapsto H(x_3)$
$x_4 \mapsto H(x_4)$
$x_5 \mapsto H(x_5)$
$x_6 \mapsto H(x_6)$

the adversary sees a few rows

$\neq$

the adversary
runs H itself:
no queries to watch

real hash function



a small circuit

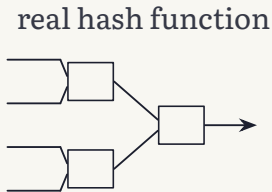
Random oracles and real hash functions

random oracle

$x_0 \mapsto H(x_0)$
$x_1 \mapsto H(x_1)$
$x_2 \mapsto H(x_2)$
$x_3 \mapsto H(x_3)$
$x_4 \mapsto H(x_4)$
$x_5 \mapsto H(x_5)$
$x_6 \mapsto H(x_6)$

the adversary sees a few rows

$\neq$
the adversary
runs H itself:
no queries to watch



a small circuit

correlation intractability · correlation robustness · correlated-input
security · UCE · extreme lossiness
[CGH98, IKNP03, GOR11, BHK13, Zha16]

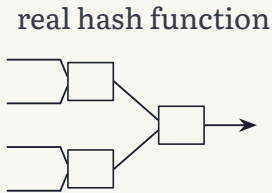
Random oracles and real hash functions

random oracle

$x_0 \mapsto H(x_0)$
$x_1 \mapsto H(x_1)$
$x_2 \mapsto H(x_2)$
$x_3 \mapsto H(x_3)$
$x_4 \mapsto H(x_4)$
$x_5 \mapsto H(x_5)$
$x_6 \mapsto H(x_6)$

the adversary sees a few rows

$\neq$
the adversary
runs H itself:
no queries to watch



a small circuit

correlation intractability · correlation robustness · correlated-input
security · UCE · **extreme lossiness**

[CGH98, IKNP03, GOR11, BHK13, Zha16]

Random oracles and real hash functions

random oracle

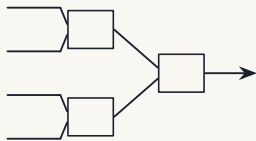
$x_0 \mapsto H(x_0)$
$x_1 \mapsto H(x_1)$
$x_2 \mapsto H(x_2)$
$x_3 \mapsto H(x_3)$
$x_4 \mapsto H(x_4)$
$x_5 \mapsto H(x_5)$
$x_6 \mapsto H(x_6)$

the adversary sees a few rows

$\neq$

the adversary
runs H itself:
no queries to watch

real hash function



a small circuit

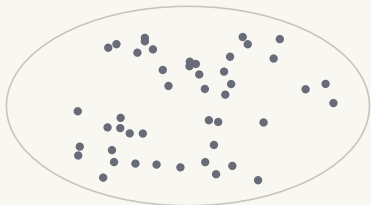
correlation intractability · correlation robustness · correlated-input
security · UCE · **extreme lossiness**

[CGH98, IKNP03, GOR11, BHK13, Zha16]



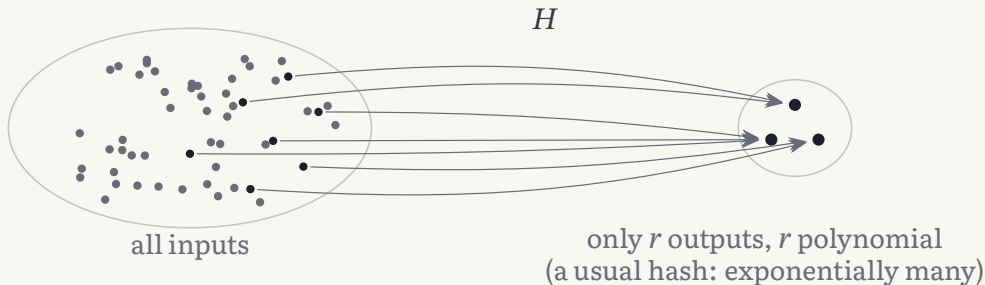
Keep an eye on that last one.

A hash with a small image

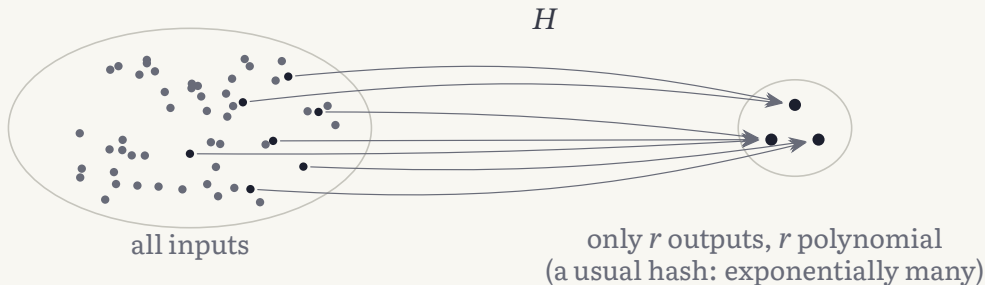


all inputs

A hash with a small image

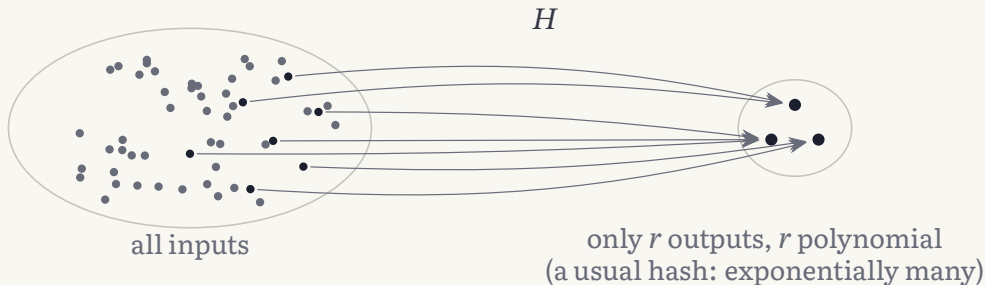


A hash with a small image



A proof often needs the hash of a secret it does not know.
With an oracle, it is a query. With r outputs, **try each**.

A hash with a small image



A proof often needs the hash of a secret it does not know.
With an oracle, it is a query. With r outputs, **try each**.



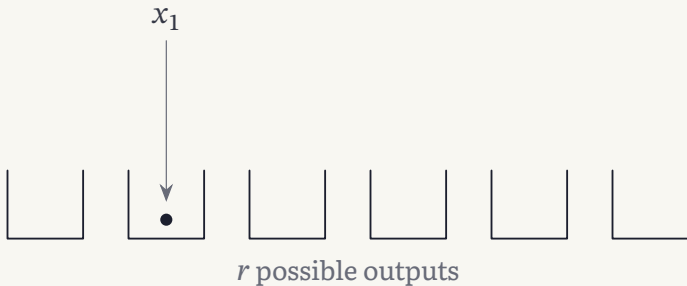
Collisions everywhere: a strange thing to want, I admit.

A small image is easy to detect

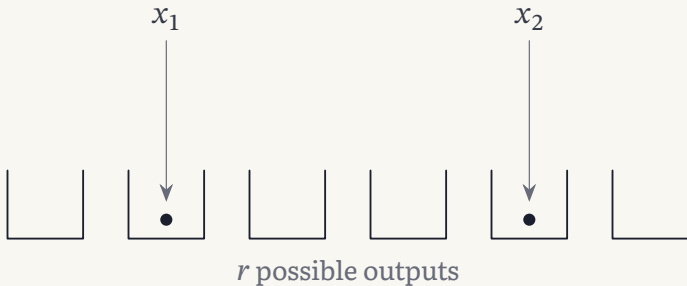


r possible outputs

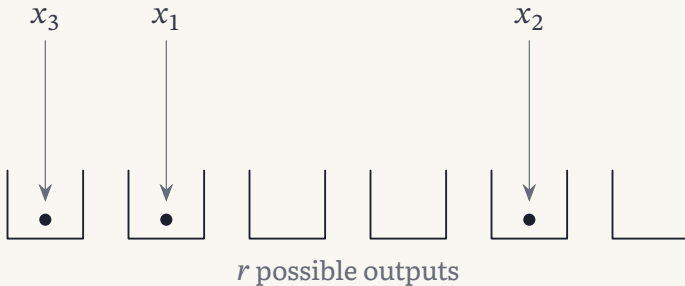
A small image is easy to detect



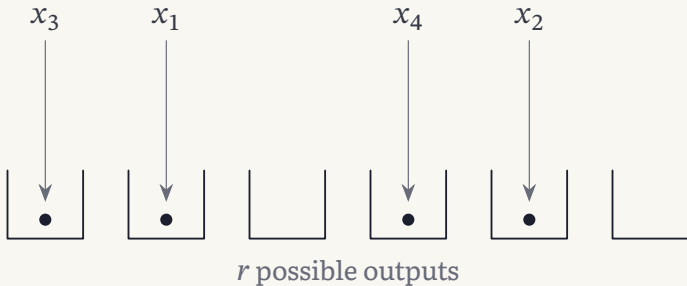
A small image is easy to detect



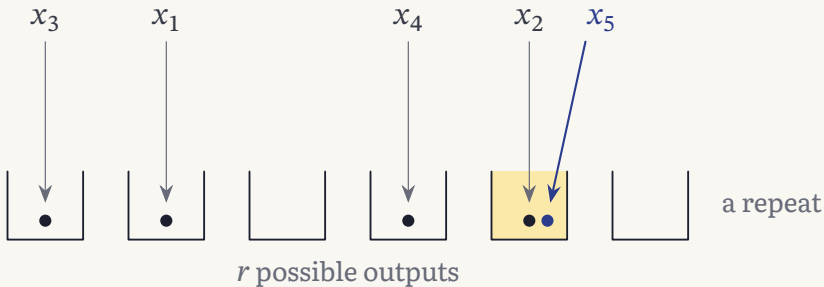
A small image is easy to detect



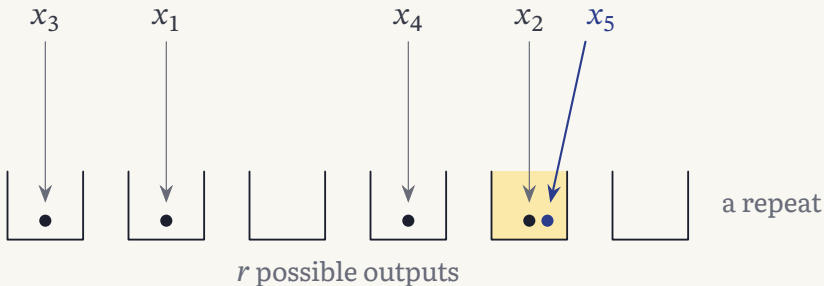
A small image is easy to detect



A small image is easy to detect



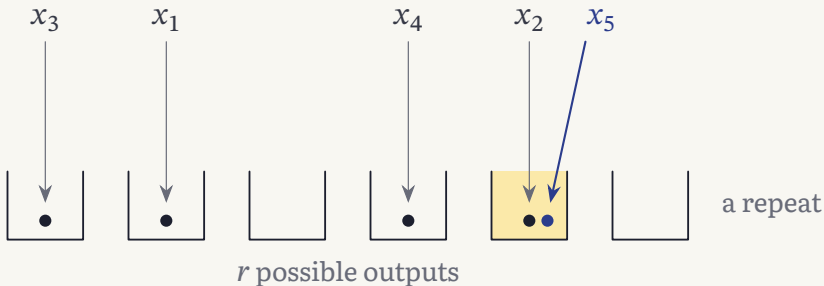
A small image is easy to detect



Injective: distinct inputs never repeat.

At most r outputs: a repeat after about $\sqrt{r}$ random inputs.

A small image is easy to detect



Injective: distinct inputs never repeat.

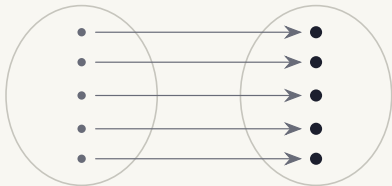
At most r outputs: a repeat after about $\sqrt{r}$ random inputs.



The birthday paradox, on the attacker's side as usual.

Extremely lossy functions

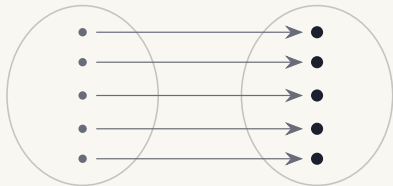
[Zha16]



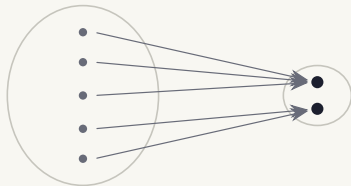
injective mode

Extremely lossy functions

[Zha16]



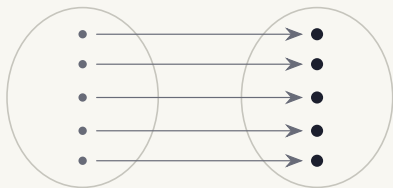
injective mode



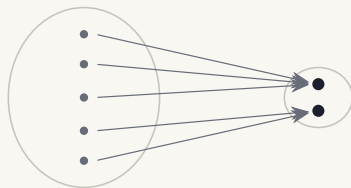
lossy mode: at most r outputs

Extremely lossy functions

[Zha16]



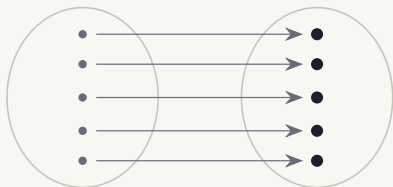
injective mode



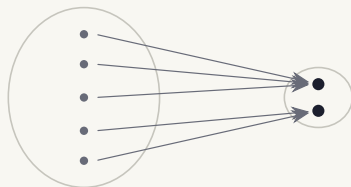
lossy mode: at most r outputs

Extremely lossy functions

[Zha16]



injective mode

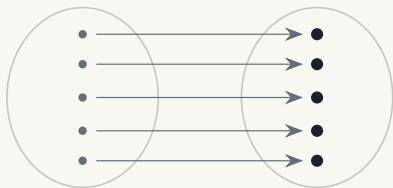


lossy mode: at most r outputs

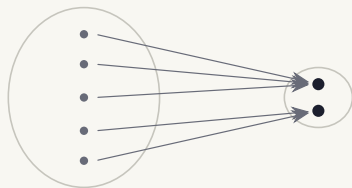
With r large enough, a fixed-size observer cannot tell the keys apart.

Extremely lossy functions

[Zha16]



injective mode



lossy mode: at most r outputs

With r large enough, a fixed-size observer cannot tell the keys apart.



That's me! An ELF. Which mode am I in? You can't tell.

Choosing r after the observer

[Zha16]



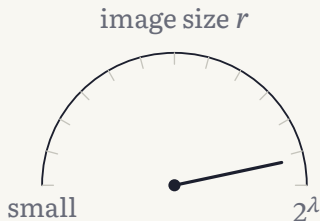
observer of size t ,
target advantage $1/\delta$

Choosing r after the observer

[Zha16]

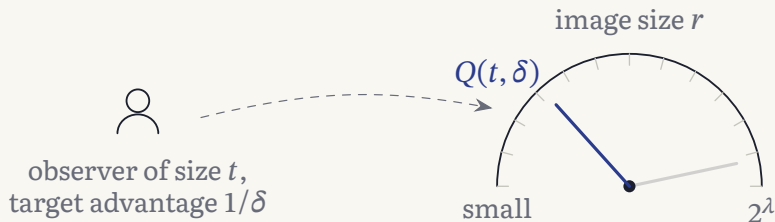


observer of size t ,
target advantage $1/\delta$



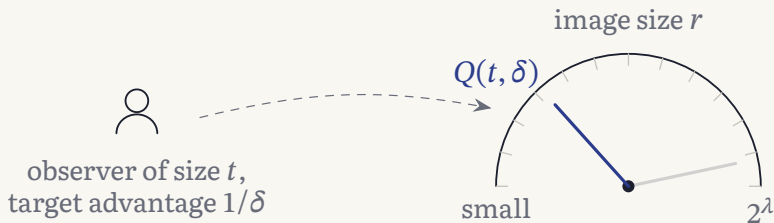
Choosing r after the observer

[Zha16]



Choosing r after the observer

[Zha16]



yes: $\forall \mathcal{A} \exists r$

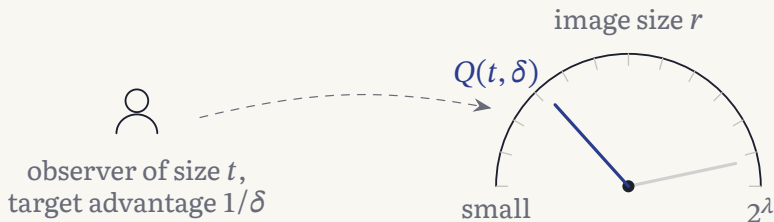
for each observer, some polynomial r holds it below $1/\delta$

no: $\exists r \forall \mathcal{A}$

one polynomial r that does so for every observer

Choosing r after the observer

[Zha16]



yes: $\forall \mathcal{A} \exists r$

for each observer, some polynomial r holds it below $1/\delta$

no: $\exists r \forall \mathcal{A}$

one polynomial r that does so for every observer



Tell me your size first. Then I choose my r .

Why a small r helps

[Zha16]

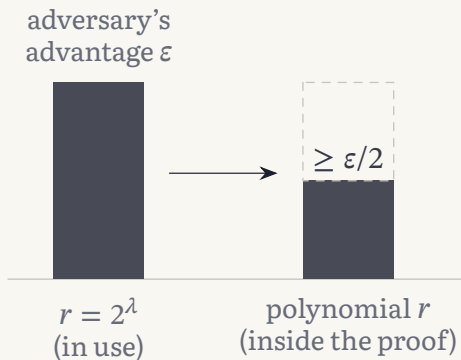
adversary's
advantage ε



$r = 2^\lambda$
(in use)

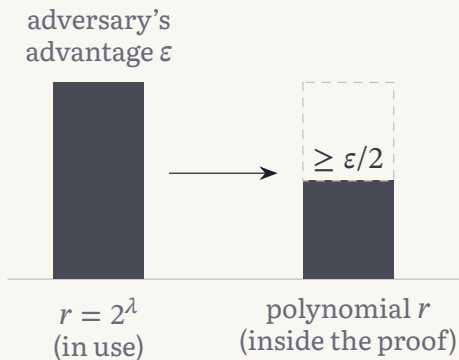
Why a small r helps

[Zha16]



Why a small r helps

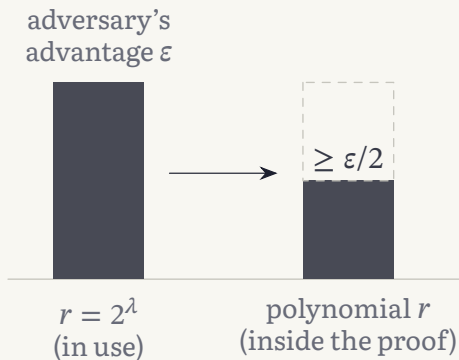
[Zha16]



the reduction runs the adversary and guesses which of the r outputs it needs: right w.p. $1/r$

Why a small r helps

[Zha16]

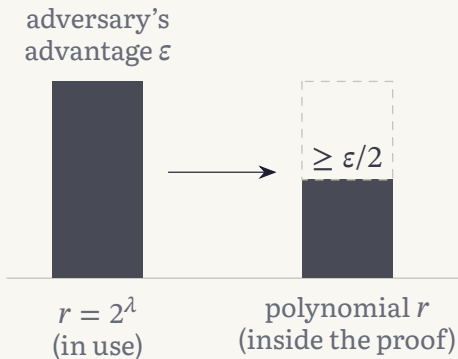


the reduction runs the adversary and guesses which of the r outputs it needs: right w.p. $1/r$

$r = 2^\lambda$: it breaks the assumption with only $\approx \epsilon/2^\lambda$: no contradiction

Why a small r helps

[Zha16]



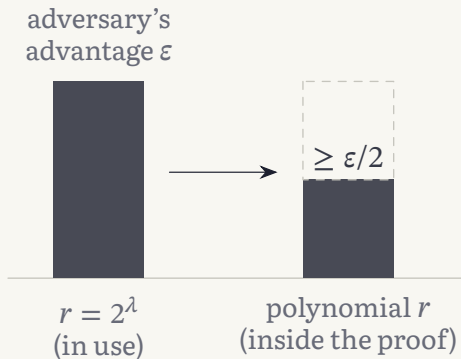
the reduction runs the adversary and guesses which of the r outputs it needs: right w.p. $1/r$

$r = 2^\lambda$: it breaks the assumption with only $\approx \epsilon/2^\lambda$: no contradiction

polynomial r : it breaks it with $\approx \epsilon/(2r)$: a contradiction

Why a small r helps

[Zha16]



the reduction runs the adversary and guesses which of the r outputs it needs: right w.p. $1/r$

$r = 2^\lambda$: it breaks the assumption with only $\approx \varepsilon/2^\lambda$: no contradiction

polynomial r : it breaks it with $\approx \varepsilon/(2r)$: a contradiction



Half the advantage, for a polynomial loss. A good trade.

Applications

from a plain ELF

point obfuscation with auxiliary input

leakage resilience

output-intractable hashing

[Zha16]

Applications

from a plain ELF

point obfuscation with auxiliary input

leakage resilience

output-intractable hashing

[Zha16]

from variants of the definition

deterministic encryption

correlated-input password hashing

anamorphic-resistant encryption

[Zha19, ABG+26]

Applications

from a plain ELF

point obfuscation with auxiliary input
leakage resilience
output-intractable hashing

[Zha16]

from variants of the definition

deterministic encryption
correlated-input password hashing
anamorphic-resistant encryption

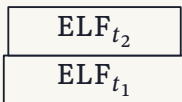
[Zha19, ABG+26]



Every new application seems to want a new variant of me.

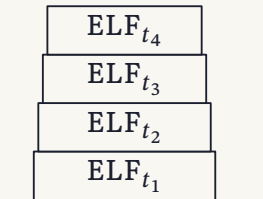
Prior constructions

[Zha16, Zha19, FGK+10]



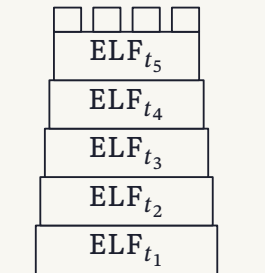
Prior constructions

[Zha16, Zha19, FGK+10]



Prior constructions

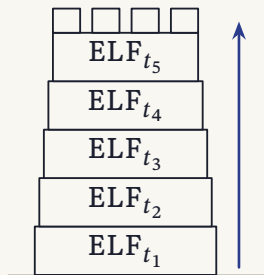
[Zha16, Zha19, FGK+10]



storey i : an ELF safe only against running time t_i
its group size fixes its security: one storey per running time

Prior constructions

[Zha16, Zha19, FGK+10]

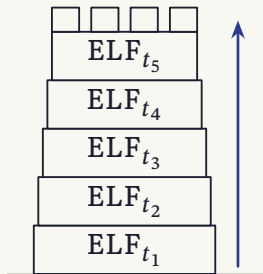


storey i : an ELF safe only against running time t_i
its group size fixes its security: one storey per running time

combiner: the input passes through every storey
→ safe against every polynomial running time

Prior constructions

[Zha16, Zha19, FGK+10]



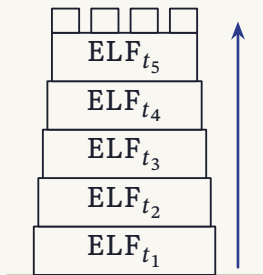
storey i : an ELF safe only against running time t_i
its group size fixes its security: one storey per running time

combiner: the input passes through every storey
→ safe against every polynomial running time

why this hurts  $\int \not\rightarrow \times \rightarrow \bigcirc \langle$
each storey needs exponential DDH: only elliptic curves
complicated, inefficient, and homomorphism is hard to expose

Prior constructions

[Zha16, Zha19, FGK+10]



storey i : an ELF safe only against running time t_i
its group size fixes its security: one storey per running time

combiner: the input passes through every storey
→ safe against every polynomial running time

why this hurts  $\int \rightarrow \times$ $\circ$ $($
each storey needs exponential DDH: only elliptic curves
complicated, inefficient, and homomorphism is hard to expose



Every new variant of me meant rebuilding the whole tower.

Our results

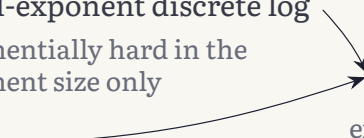
small-exponent discrete log
exponentially hard in the
exponent size only

DCR
polynomially hard

Our results

small-exponent discrete log
exponentially hard in the
exponent size only

DCR
polynomially hard



ELTDH

extremely lossy
trapdoor hashing

Our results



Our results



Every variant of me, in one go. No tower.

ASSUMPTION

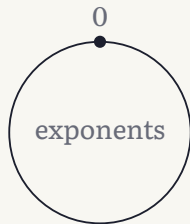


Only one new assumption in this whole talk. Here it is.

Small exponents

[vOW96, PS98]

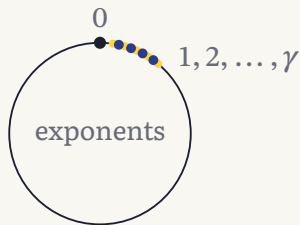
$\mathbb{G} = \langle g \rangle$: cyclic, g public, size $\geq 2^\lambda$ (ours, later: mod N^2 , N an RSA modulus)



Small exponents

[vOW96, PS98]

$\mathbb{G} = \langle g \rangle$: cyclic, g public, size $\geq 2^\lambda$ (ours, later: mod N^2 , N an RSA modulus)

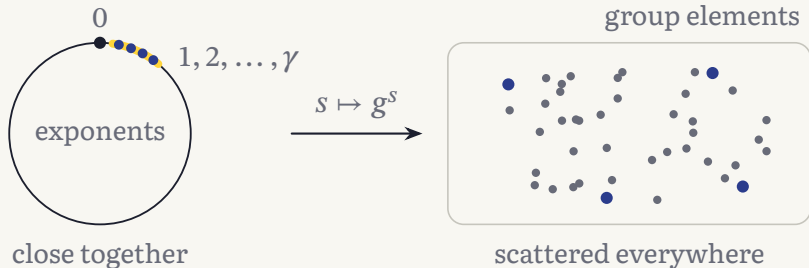


close together

Small exponents

[vOW96, PS98]

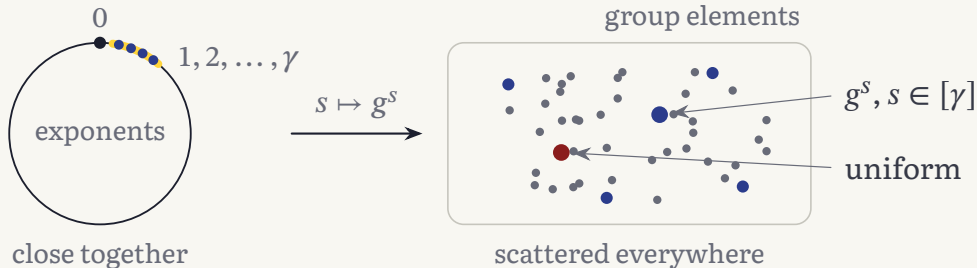
$\mathbb{G} = \langle g \rangle$: cyclic, g public, size $\geq 2^\lambda$ (ours, later: mod N^2 , N an RSA modulus)



Small exponents

[vOW96, PS98]

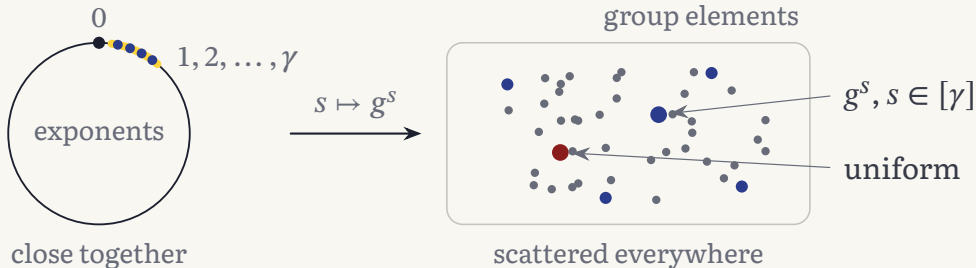
$\mathbb{G} = \langle g \rangle$: cyclic, g public, size $\geq 2^\lambda$ (ours, later: mod N^2 , N an RSA modulus)



Small exponents

[vOW96, PS98]

$\mathbb{G} = \langle g \rangle$: cyclic, g public, size $\geq 2^\lambda$ (ours, later: mod N^2 , N an RSA modulus)



DSEDL: hard to tell apart once γ is large enough.

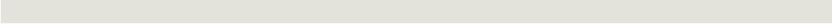


Close in the exponent, scattered in the group.

Choosing γ after the observer

[Sha71]

a fixed interval $[\gamma]$, and an observer with T steps

 γ

Choosing γ after the observer

[Sha71]

a fixed interval $[\gamma]$, and an observer with T steps



Choosing γ after the observer

[Sha71]

a fixed interval $[\gamma]$, and an observer with T steps



advantage about T^2/γ : every fixed polynomial γ is broken

Choosing γ after the observer

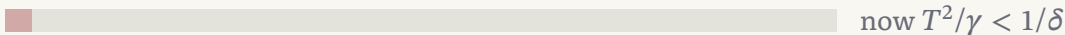
[Sha71]

a fixed interval $[\gamma]$, and an observer with T steps



advantage about T^2/γ : every **fixed** polynomial γ is broken

so choose $\gamma \geq Q(t, \delta)$ **after** the observer



Choosing γ after the observer

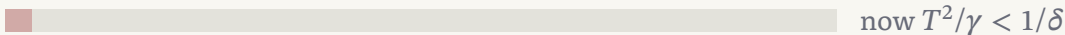
[Sha71]

a fixed interval $[\gamma]$, and an observer with T steps



advantage about T^2/γ : every **fixed** polynomial γ is broken

so choose $\gamma \geq Q(t, \delta)$ **after** the observer



Generic attacks provably do no better; DSEDL assumes no attack does.

Choosing γ after the observer

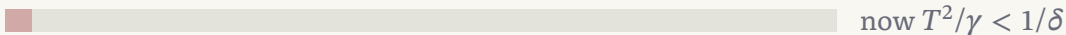
[Sha71]

a fixed interval $[\gamma]$, and an observer with T steps



advantage about T^2/γ : every **fixed** polynomial γ is broken

so choose $\gamma \geq Q(t, \delta)$ **after** the observer



Generic attacks provably do no better; DSEDL assumes no attack does.



Shanks, 1971. Still the best attack we know of here.

Group size and exponent size

exponential DDH



$\log |\mathbb{G}|$ bits

Group size and exponent size

exponential DDH

$\log |\mathbb{G}|$ bits

attacks must cost $2^{\Omega(\log |\mathbb{G}|)}$; index calculus beats that, except on elliptic curves

Group size and exponent size

exponential DDH

$\log |\mathbb{G}|$ bits

attacks must cost $2^{\Omega(\log |\mathbb{G}|)}$; index calculus beats that, except on elliptic curves

tunable DSEDH

$\log \gamma$ bits

$\log |\mathbb{G}|$ bits

Group size and exponent size

exponential DDH

 $\log |\mathbb{G}|$ bits

attacks must cost $2^{\Omega(\log |\mathbb{G}|)}$; index calculus beats that, except on elliptic curves

tunable DSEDL

 $\log \gamma$ bits

 $\log |\mathbb{G}|$ bits

attacks must cost $2^{\Omega(\log \gamma)}$, about $\sqrt{\gamma}$; a large group stops index calculus

Group size and exponent size

exponential DDH

 $\log |\mathbb{G}|$ bits

attacks must cost $2^{\Omega(\log |\mathbb{G}|)}$; index calculus beats that, except on elliptic curves

tunable DSEDL

 $\log \gamma$ bits

$\log |\mathbb{G}|$ bits

attacks must cost $2^{\Omega(\log \gamma)}$, about $\sqrt{\gamma}$; a large group stops index calculus

So it plausibly holds modulo N^2 , and in many other groups.

Group size and exponent size

exponential DDH

$\log |\mathbb{G}|$ bits

attacks must cost $2^{\Omega(\log |\mathbb{G}|)}$; index calculus beats that, except on elliptic curves

tunable DSED

$\log \gamma$ bits

$\log |\mathbb{G}|$ bits

attacks must cost $2^{\Omega(\log \gamma)}$, about $\sqrt{\gamma}$; a large group stops index calculus

So it plausibly holds modulo N^2 , and in many other groups.



Tuning γ needs no new group. Goodbye, tower.

CONSTRUCTION



Two small changes to a known construction. Watch closely.

Trapdoor hashing

[DGI+19]

public hash key hk



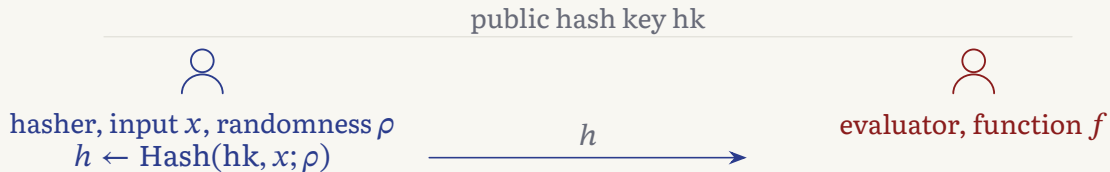
hasher, input x , randomness ρ



evaluator, function f

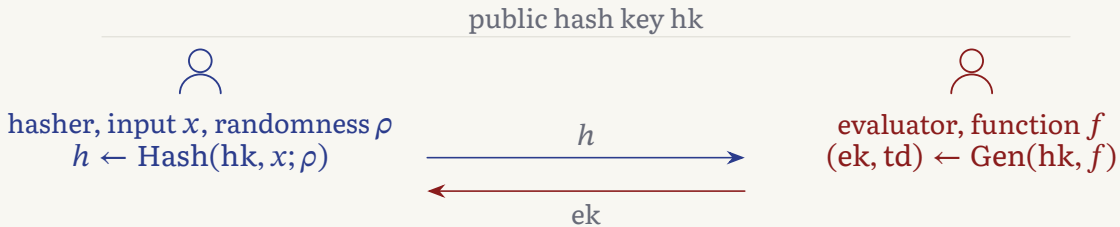
Trapdoor hashing

[DGI+19]



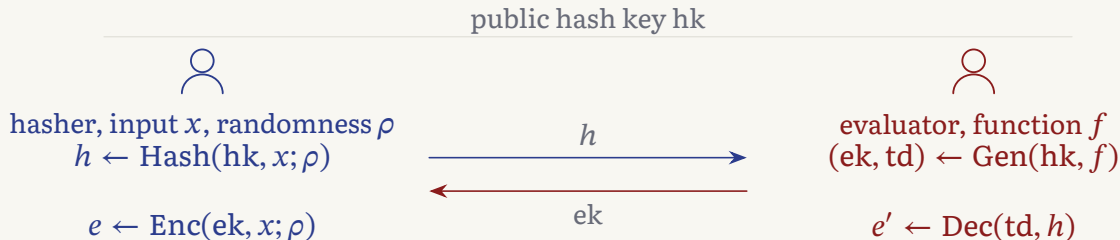
Trapdoor hashing

[DGI+19]



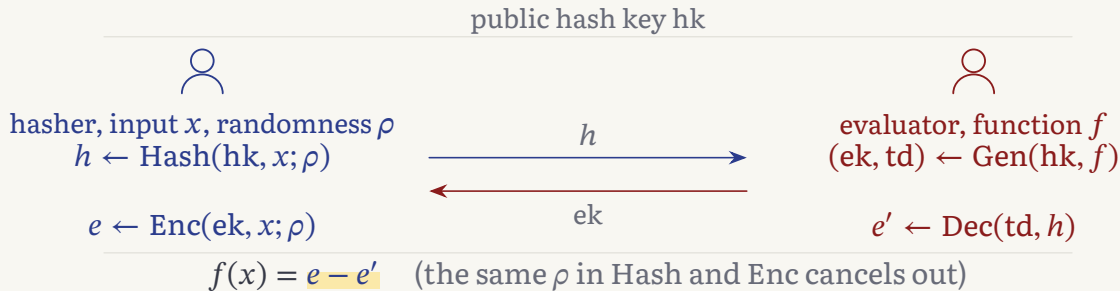
Trapdoor hashing

[DGI+19]



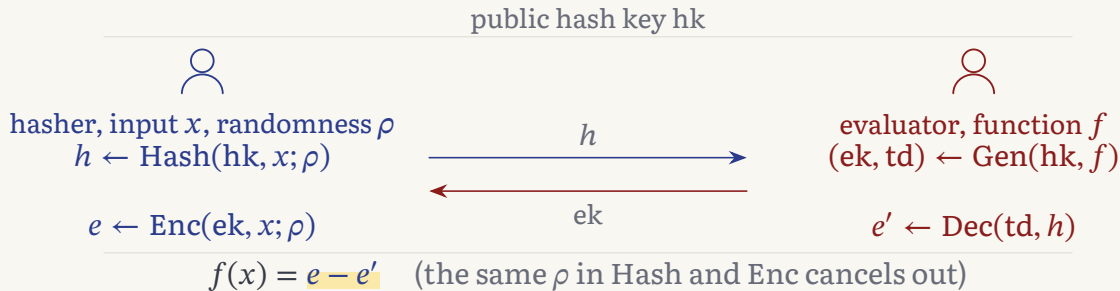
Trapdoor hashing

[DGI+19]



Trapdoor hashing

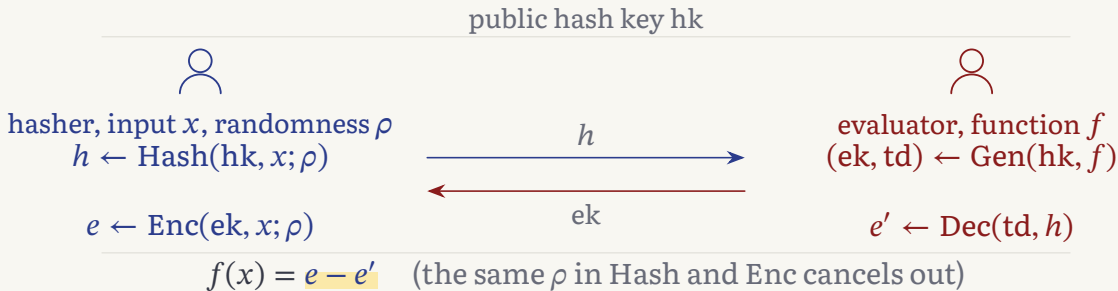
[DGI+19]



h hides x , ek hides f ; with td , (h, e) gives $f(x)$.

Trapdoor hashing

[DGI+19]

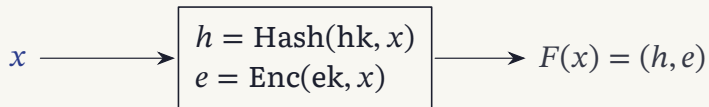


h hides x , ek hides f ; with td , (h, e) gives $f(x)$.

A hash that answers one question about its input.

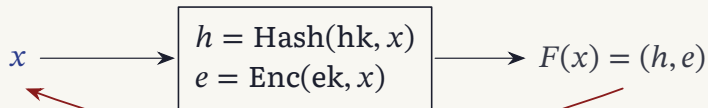
A trapdoor function inside

key (hk, ek) , f the identity, and $\rho = 0$: same x , same output



A trapdoor function inside

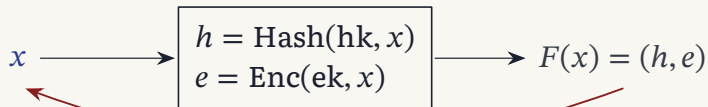
key (hk, ek) , f the identity, and $\rho = 0$: same x , same output



$\bigcirc_{\pi} e - \text{Dec}(td, h) = x$: injective, and td inverts it

A trapdoor function inside

key (hk, ek) , f the identity, and $\rho = 0$: same x , same output



$\bigcirc_{\pi} e - \text{Dec}(\text{td}, h) = x$: injective, and td inverts it

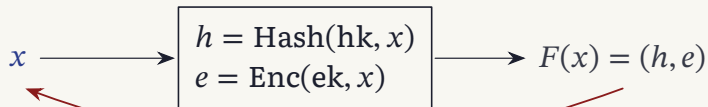
missing: a lossy mode, with few outputs (h, e)

plain trapdoor hashing keeps many outputs, because a repeat would leak f :

pick x, x' : $f_0(x) = f_0(x')$, $f_1(x) \neq f_1(x')$; equal $(h, e) \Rightarrow ek$ encodes f_0 , not f_1

A trapdoor function inside

key (hk, ek) , f the identity, and $\rho = 0$: same x , same output



$\bigcirc_{\pi} e - \text{Dec}(\text{td}, h) = x$: injective, and td inverts it

missing: a lossy mode, with few outputs (h, e)

plain trapdoor hashing keeps many outputs, because a repeat would leak f :
pick x, x' : $f_0(x) = f_0(x')$, $f_1(x) \neq f_1(x')$; equal $(h, e) \Rightarrow \text{ek}$ encodes f_0 , not f_1

ELTDH: setup takes r ; $\leq r$ digests at $\rho = 0$; $r \geq Q(t, \delta)$ passes as full

The Damgård–Jurik group

[Pai99, DJ01]

$N = pq$ with safe primes: $p = 2p' + 1$ and $q = 2q' + 1$, with p', q' prime
mod N^2 we use products $c \cdot \exp(x)$: a hard part times an easy part

hard part $c \in \mathbb{G}_N$
discrete log hard

$\mathbb{G}_N = \{z^{2N}\}$: the squares of the N -th residues, cyclic of order $p'q'$

The Damgård–Jurik group

[Pai99, DJ01]

$N = pq$ with safe primes: $p = 2p' + 1$ and $q = 2q' + 1$, with p', q' prime
mod N^2 we use products $c \cdot \exp(x)$: a hard part times an easy part

hard part $c \in \mathbb{G}_N$ discrete log hard	easy part $\exp(x)$ discrete log easy	$\exp(x) = 1 + Nx$ $\log(1 + Nx) = x$
---	--	--

$\mathbb{G}_N = \{z^{2N}\}$: the squares of the N -th residues, cyclic of order $p'q'$

$\exp(a + b) = \exp(a) \exp(b)$: adding in $\mathbb{Z}_N$ becomes multiplying mod N^2

The Damgård–Jurik group

[Pai99, DJ01]

$N = pq$ with safe primes: $p = 2p' + 1$ and $q = 2q' + 1$, with p', q' prime
mod N^2 we use products $c \cdot \exp(x)$: a hard part times an easy part

hard part $c \in \mathbb{G}_N$ discrete log hard	easy part $\exp(x)$ discrete log easy	$\exp(x) = 1 + Nx$ $\log(1 + Nx) = x$
---	--	--

$\mathbb{G}_N = \{z^{2N}\}$: the squares of the N -th residues, cyclic of order $p'q'$

$\exp(a + b) = \exp(a) \exp(b)$: adding in $\mathbb{Z}_N$ becomes multiplying mod N^2

DCR: a random hard part c looks like a random $c \cdot \exp(x)$

so multiplying by a random hard part hides any easy part $\exp(y)$

The Damgård–Jurik group

[Pai99, DJ01]

$N = pq$ with safe primes: $p = 2p' + 1$ and $q = 2q' + 1$, with p', q' prime
mod N^2 we use products $c \cdot \exp(x)$: a hard part times an easy part

hard part $c \in \mathbb{G}_N$
discrete log hard

easy part $\exp(x)$
discrete log easy

$\exp(x) = 1 + Nx$
 $\log(1 + Nx) = x$

$\mathbb{G}_N = \{z^{2N}\}$: the squares of the N -th residues, cyclic of order $p'q'$

$\exp(a + b) = \exp(a) \exp(b)$: adding in $\mathbb{Z}_N$ becomes multiplying mod N^2

DCR: a random hard part c looks like a random $c \cdot \exp(x)$

so multiplying by a random hard part hides any easy part $\exp(y)$



Safe primes: assumed plentiful. Infinitely many? Still unproved.

Distributed discrete log

[RS21, OSY21]

MULTIPLICATIVE SHARES $\rightarrow$ ADDITIVE SHARES

 holds $c \cdot \exp(x)$  holds c $\longrightarrow$ values $e, e' \in \mathbb{Z}_N$ with $e - e' = x$,
quotient: $c \exp(x) / c = \exp(x)$ without talking

Distributed discrete log

[RS21, OSY21]

MULTIPLICATIVE SHARES $\rightarrow$ ADDITIVE SHARES

 holds $c \cdot \exp(x)$  holds c $\longrightarrow$ values $e, e' \in \mathbb{Z}_N$ with $e - e' = x$,
quotient: $c \exp(x)/c = \exp(x)$ without talking

why: the hasher will hold $h^\sigma \exp(f(x))$ and the evaluator h^σ , so $e - e' = f(x)$

Distributed discrete log

[RS21, OSY21]

MULTIPLICATIVE SHARES $\rightarrow$ ADDITIVE SHARES

 holds $c \cdot \exp(x)$  holds c $\longrightarrow$ values $e, e' \in \mathbb{Z}_N$ with $e - e' = x$,
quotient: $c \exp(x)/c = \exp(x)$ without talking

why: the hasher will hold $h^\sigma \exp(f(x))$ and the evaluator h^σ , so $e - e' = f(x)$

THE TRICK

let $w := c \bmod N$; as $\exp(x) = 1 + Nx \equiv 1$, also $c \cdot \exp(x) \equiv w \pmod{N}$

$c \cdot w^{-1} \equiv 1 \pmod{N}$, so it lies in $1 + N\mathbb{Z}$, where log is easy

Distributed discrete log

[RS21, OSY21]

MULTIPLICATIVE SHARES $\rightarrow$ ADDITIVE SHARES

 holds $c \cdot \exp(x)$  holds c $\longrightarrow$ values $e, e' \in \mathbb{Z}_N$ with $e - e' = x$,
quotient: $c \exp(x)/c = \exp(x)$ without talking

why: the hasher will hold $h^\sigma \exp(f(x))$ and the evaluator h^σ , so $e - e' = f(x)$

THE TRICK

let $w := c \bmod N$; as $\exp(x) = 1 + Nx \equiv 1$, also $c \cdot \exp(x) \equiv w \pmod{N}$

$c \cdot w^{-1} \equiv 1 \pmod{N}$, so it lies in $1 + N\mathbb{Z}$, where log is easy

$\text{DDLog}(c) := \log(c w^{-1})$, and $\log(c w^{-1} \exp(x)) = \log(c w^{-1}) + x$

Distributed discrete log

[RS21, OSY21]

MULTIPLICATIVE SHARES $\rightarrow$ ADDITIVE SHARES

 holds $c \cdot \exp(x)$  holds c $\longrightarrow$ values $e, e' \in \mathbb{Z}_N$ with $e - e' = x$,
quotient: $c \exp(x)/c = \exp(x)$ without talking

why: the hasher will hold $h^\sigma \exp(f(x))$ and the evaluator h^σ , so $e - e' = f(x)$

THE TRICK

let $w := c \bmod N$; as $\exp(x) = 1 + Nx \equiv 1$, also $c \cdot \exp(x) \equiv w \pmod{N}$

$c \cdot w^{-1} \equiv 1 \pmod{N}$, so it lies in $1 + N\mathbb{Z}$, where log is easy

$\text{DDLog}(c) := \log(c w^{-1})$, and $\log(c w^{-1} \exp(x)) = \log(c w^{-1}) + x$



Multiplying by $\exp(x)$ in the group becomes adding x to the values.

Trapdoor hashing from groups

[DGI+19, RS21]

HASH KEY

• • • •

$g_0, g_1, \dots, g_d$

digest $h = g_0^{\rho} \prod_{i=1}^d g_i^{x_i}$

Trapdoor hashing from groups

[DGI+19, RS21]

HASH KEY $\bullet \bullet \bullet \bullet g_0, g_1, \dots, g_d$ digest $h = g_0^\rho \prod_{i=1}^d g_i^{x_i}$

ENCODING KEY $K_i = \boxed{g_i^\sigma} \boxed{\exp(y_i)}$ and $K_0 = g_0^\sigma \circ_{\pi} \text{td} = \sigma$

Trapdoor hashing from groups

[DGI+19, RS21]

HASH KEY • • • • $g_0, g_1, \dots, g_d$ digest $h = g_0^\rho \prod_{i=1}^d g_i^{x_i}$

ENCODING KEY $K_i =$

g_i^σ	$\exp(y_i)$
--------------	-------------

 and $K_0 = g_0^\sigma$ $\text{O-}\pi$ $\text{td} = \sigma$



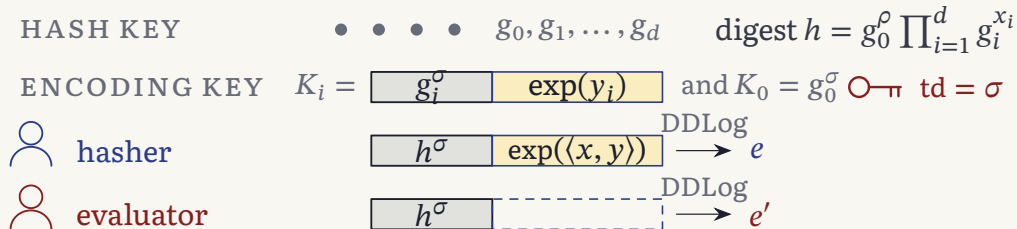
hasher

h^σ	$\exp(\langle x, y \rangle)$
------------	------------------------------

 $\xrightarrow{\text{DDLog}}$ e

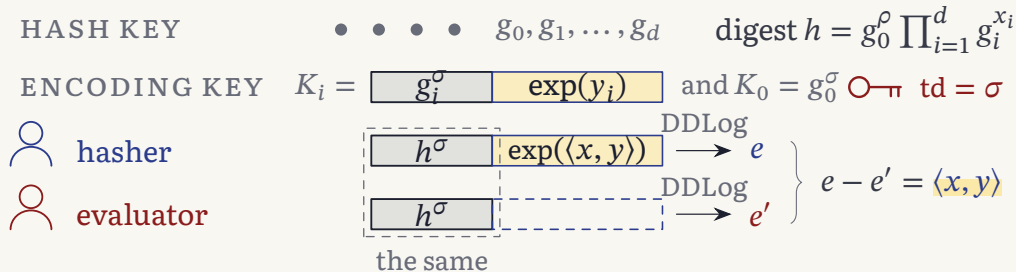
Trapdoor hashing from groups

[DGI+19, RS21]



Trapdoor hashing from groups

[DGI+19, RS21]



input $x \in \mathbb{Z}_N^d$, linear function $f(x) = \langle x, y \rangle$

$K_0^\rho \prod_i K_i^{x_i} = h^\sigma \exp(\langle x, y \rangle)$: the **same** ρ makes the hard parts match.

g_0^ρ hides x in h ; g_i^σ hides $\exp(y_i)$ in ek (DCR).

The construction

WHAT IS HASHED

THE BASES

The construction

WHAT IS HASHED

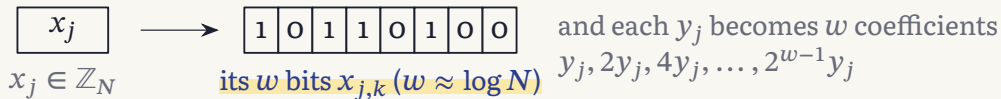


$x_j \in \mathbb{Z}_N$ its w bits $x_{j,k}$ ($w \approx \log N$)

THE BASES

The construction

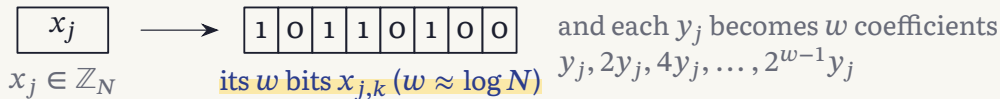
WHAT IS HASHED



THE BASES

The construction

WHAT IS HASHED



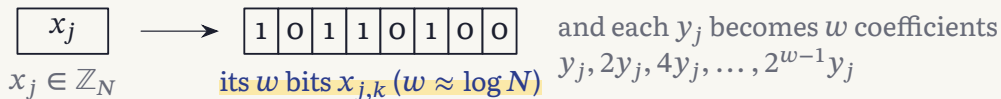
THE BASES



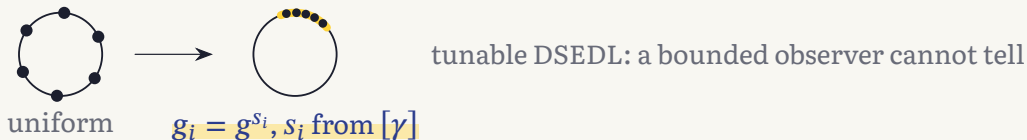
uniform

The construction

WHAT IS HASHED

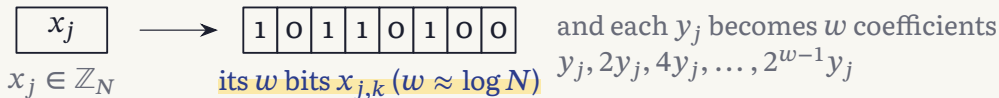


THE BASES

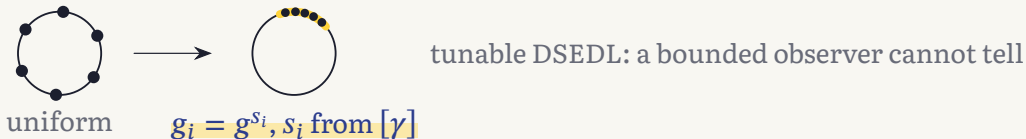


The construction

WHAT IS HASHED



THE BASES



Same answer: $\sum_k (2^k y_j) x_{j,k} = y_j \sum_k 2^k x_{j,k} = y_j x_j$.

Why bits: the digest's exponent $\sum_i s_i x_i$ is then a short sum of small values.

Bounding the image

$$h = \prod_{i=1}^n g_i^{x_i} = g^{\sum_i s_i x_i}, \quad n = dw \text{ bits } x_i, \quad s_i \leq \gamma, \quad \rho = 0 \text{ (as in the ELF)}$$

Bounding the image

$$h = \prod_{i=1}^n g_i^{x_i} = g^{\sum_i s_i x_i}, \quad n = dw \text{ bits } x_i, \quad s_i \leq \gamma, \quad \rho = 0 \text{ (as in the ELF)}$$



Bounding the image

$$h = \prod_{i=1}^n g_i^{x_i} = g^{\sum_i s_i x_i}, \quad n = dw \text{ bits } x_i, \quad s_i \leq \gamma, \quad \rho = 0 \text{ (as in the ELF)}$$

every digest is one of these: for every key, no assumption



Bounding the image

$$h = \prod_{i=1}^n g_i^{x_i} = g^{\sum_i s_i x_i}, \quad n = dw \text{ bits } x_i, \quad s_i \leq \gamma, \quad \rho = 0 \text{ (as in the ELF)}$$

every digest is one of these: for every key, no assumption



Bounding the image

$$h = \prod_{i=1}^n g_i^{x_i} = g^{\sum_i s_i x_i}, \quad n = dw \text{ bits } x_i, \quad s_i \leq \gamma, \quad \rho = 0 \text{ (as in the ELF)}$$

every digest is one of these: for every key, no assumption



With $\gamma = \lfloor (r-1)/n \rfloor$: at most $n\gamma + 1 \leq r$ digests.

Bounding the image

$$h = \prod_{i=1}^n g_i^{x_i} = g^{\sum_i s_i x_i}, \quad n = dw \text{ bits } x_i, \quad s_i \leq \gamma, \quad \rho = 0 \text{ (as in the ELF)}$$

every digest is one of these: for every key, no assumption



With $\gamma = \lfloor (r-1)/n \rfloor$: at most $n\gamma + 1 \leq r$ digests.

$e = f(x) + \text{Dec}(\text{td}, h)$: for each value of $f(x)$, at most r pairs (h, e) .

Bounding the image

$$h = \prod_{i=1}^n g_i^{x_i} = g^{\sum_i s_i x_i}, \quad n = dw \text{ bits } x_i, \quad s_i \leq \gamma, \quad \rho = 0 \text{ (as in the ELF)}$$

every digest is one of these: for every key, no assumption



With $\gamma = \lfloor (r-1)/n \rfloor$: at most $n\gamma + 1 \leq r$ digests.

$e = f(x) + \text{Dec}(\text{td}, h)$: for each value of $f(x)$, at most r pairs (h, e) .



One knob, γ , sets both my image size and my security.

Mode indistinguishability

injective ELF key

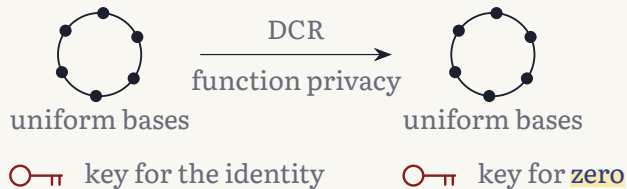


uniform bases

$\bigcirc \dashv \pi$ key for the identity

Mode indistinguishability

injective ELF key



Mode indistinguishability

injective ELF key



uniform bases

 key for the identity

DCR
function privacy



uniform bases

 key for zero

tunable DSEDL
all bases at once



small bases

 key for zero

lossy ELF key

First hide which function ek encodes, then shrink the bases.

Mode indistinguishability

injective ELF key



 π key for the identity

DCR
function privacy



 π key for zero

tunable DSEDL
all bases at once



 π key for zero

First hide which function ek encodes, then shrink the bases.

Order matters: with small bases, ek would give the function away.



APPLICATIONS



What does one hash function buy? Every variant of me.

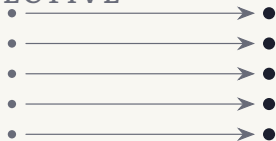
Trapdoor ELF

domain size m : input $u \in \{1, \dots, m\} \rightarrow x = u - 1 \in \{0, \dots, m - 1\} \subset \mathbb{Z}_N \rightarrow (h, e)$

Trapdoor ELF

domain size m : input $u \in \{1, \dots, m\} \rightarrow x = u - 1 \in \{0, \dots, m - 1\} \subset \mathbb{Z}_N \rightarrow (h, e)$

INJECTIVE



$$\bigcirc \dashv \pi \quad x = e - \text{Dec}(\text{td}, h)$$

Trapdoor ELF

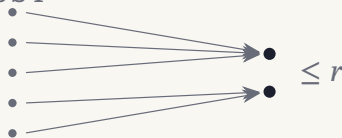
domain size m : input $u \in \{1, \dots, m\} \rightarrow x = u - 1 \in \{0, \dots, m - 1\} \subset \mathbb{Z}_N \rightarrow (h, e)$

INJECTIVE



$$\bigcirc_{\pi} \quad x = e - \text{Dec}(\text{td}, h)$$

LOSSY



zero function: $e = \text{Dec}(\text{td}, h)$, fixed by h

Trapdoor ELF

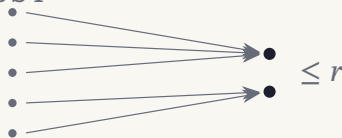
domain size m : input $u \in \{1, \dots, m\} \rightarrow x = u - 1 \in \{0, \dots, m - 1\} \subset \mathbb{Z}_N \rightarrow (h, e)$

INJECTIVE



$$\bigcirc_{\pi} \quad x = e - \text{Dec}(\text{td}, h)$$

LOSSY



zero function: $e = \text{Dec}(\text{td}, h)$, fixed by h

Trapdoor ELF

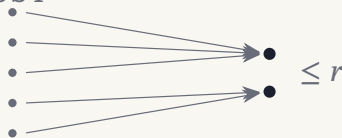
domain size m : input $u \in \{1, \dots, m\} \rightarrow x = u - 1 \in \{0, \dots, m - 1\} \subset \mathbb{Z}_N \rightarrow (h, e)$

INJECTIVE



$$\bigcirc \neg \pi \quad x = e - \text{Dec}(\text{td}, h)$$

LOSSY



zero function: $e = \text{Dec}(\text{td}, h)$, fixed by h

Key: $2w + 2$ group elements ($w \approx$ bits of N). Output: (h, e) .

Trapdoor ELF

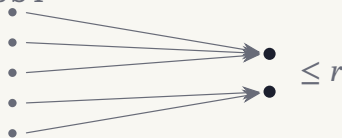
domain size m : input $u \in \{1, \dots, m\} \rightarrow x = u - 1 \in \{0, \dots, m - 1\} \subset \mathbb{Z}_N \rightarrow (h, e)$

INJECTIVE



$$\bigcirc \dashv \pi \quad x = e - \text{Dec}(\text{td}, h)$$

LOSSY



zero function: $e = \text{Dec}(\text{td}, h)$, fixed by h

Key: $2w + 2$ group elements ($w \approx$ bits of N). Output: (h, e) .



One value in $\mathbb{Z}_N$ holds my whole input: no decoding bit by bit.

Plain ELF

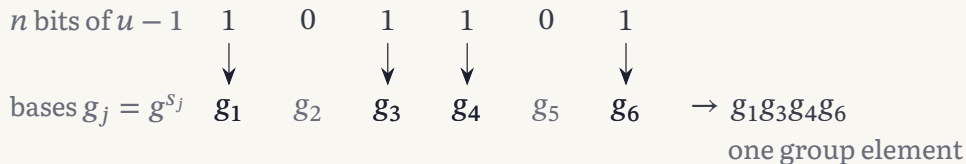
n bits of $u - 1$ 1 0 1 1 0 1

Plain ELF

n bits of $u - 1$ 1 0 1 1 0 1

bases $g_j = g^{s_j}$ g_1 g_2 g_3 g_4 g_5 g_6

Plain ELF



Plain ELF

$$\begin{array}{ccccccc}
 n \text{ bits of } u-1 & 1 & 0 & 1 & 1 & 0 & 1 \\
 & \downarrow & & \downarrow & \downarrow & & \downarrow \\
 \text{bases } g_j = g^{s_j} & g_1 & g_2 & g_3 & g_4 & g_5 & g_6
 \end{array}
 \rightarrow g_1 g_3 g_4 g_6$$

one group element

injective, s_j uniform: collision $\Rightarrow \sum_j a_j s_j \equiv 0, a \in \{0, \pm 1\}^n \setminus 0$; prob. $< 3^n / |\mathbb{G}|$

lossy: $s_j \leq \gamma$, so the output is $g^{A(u)}$ with $A(u) \in [0, n\gamma]$

Plain ELF

$$\begin{array}{cccccc}
 n \text{ bits of } u-1 & 1 & 0 & 1 & 1 & 0 & 1 \\
 & \downarrow & & \downarrow & \downarrow & & \downarrow \\
 \text{bases } g_j = g^{s_j} & g_1 & g_2 & g_3 & g_4 & g_5 & g_6
 \end{array}
 \rightarrow g_1 g_3 g_4 g_6$$

one group element

injective, s_j uniform: collision $\Rightarrow \sum_j a_j s_j \equiv 0, a \in \{0, \pm 1\}^n \setminus 0$; prob. $< 3^n/|\mathbb{G}|$

lossy: $s_j \leq \gamma$, so the output is $g^{A(u)}$ with $A(u) \in [0, n\gamma]$

No trapdoor and no DCR: **any group** where tunable DSEDL holds.

Plain ELF

$$\begin{array}{ccccccc}
 n \text{ bits of } u-1 & 1 & 0 & 1 & 1 & 0 & 1 \\
 & \downarrow & & \downarrow & \downarrow & & \downarrow \\
 \text{bases } g_j = g^{s_j} & g_1 & g_2 & g_3 & g_4 & g_5 & g_6 \rightarrow g_1 g_3 g_4 g_6 \\
 & & & & & & \text{one group element}
 \end{array}$$

injective, s_j uniform: collision $\Rightarrow \sum_j a_j s_j \equiv 0, a \in \{0, \pm 1\}^n \setminus 0$; prob. $< 3^n / |\mathbb{G}|$

lossy: $s_j \leq \gamma$, so the output is $g^{A(u)}$ with $A(u) \in [0, n\gamma]$

No trapdoor and no DCR: **any group** where tunable DSEDL holds.



My lightest version: a key of n group elements, an output of one.

SUMMARY



Time to sum up. Here is what to take home.

Summary

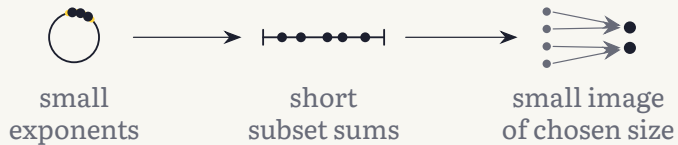


small
exponents

Summary



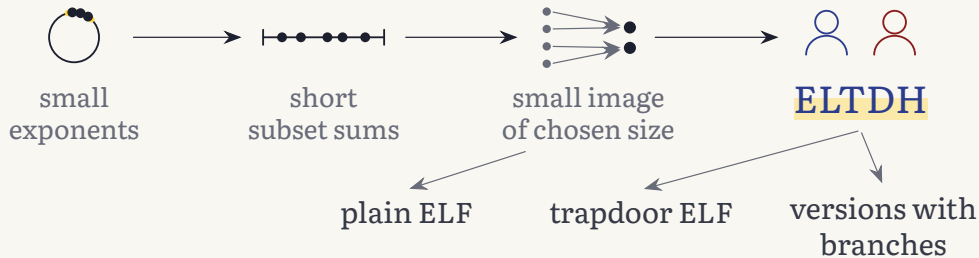
Summary



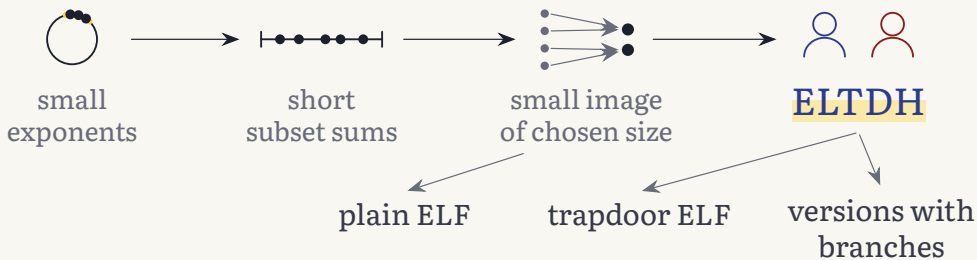
Summary



Summary

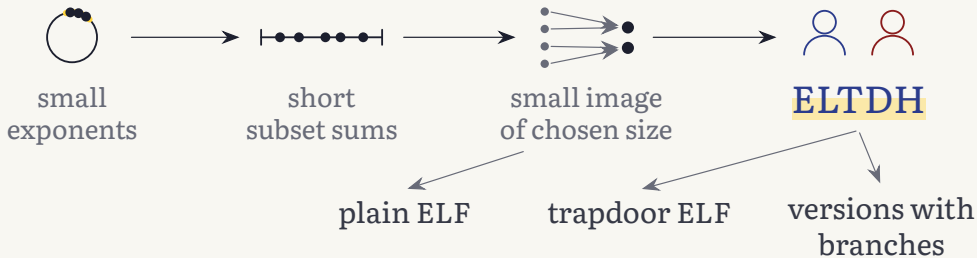


Summary



Assumptions: tunable DSEDL and DCR.
The plain ELF needs only tunable DSEDL.

Summary

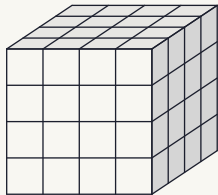


Assumptions: tunable DSEDL and DCR.
The plain ELF needs only tunable DSEDL.



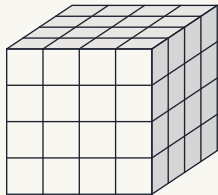
No more tower. Just one dial: the size of the exponents.

Key size

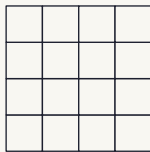


trapdoor ELF [Zha19]
 $n^{3+o(1)}$ group elements

Key size

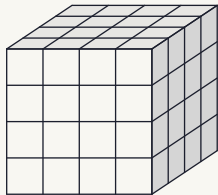


trapdoor ELF [Zha19]
 $n^{3+o(1)}$ group elements

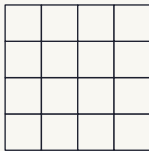


ELF [Zha16]
 $\Theta(n^2)$ group elements

Key size



trapdoor ELF [Zha19]
 $n^{3+o(1)}$ group elements

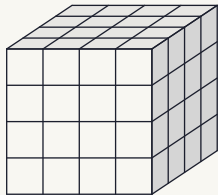


ELF [Zha16]
 $\Theta(n^2)$ group elements

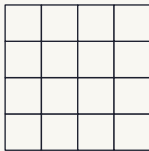


our plain ELF
 n group elements

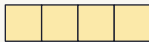
Key size



trapdoor ELF [Zha19]
 $n^{3+o(1)}$ group elements



ELF [Zha16]
 $\Theta(n^2)$ group elements



our plain ELF
 n group elements

Our trapdoor ELF: $2w + 2$ group elements.



Drawn for $n = 4$ input bits. The gap only grows from there.

Thank you

Simple Extremely Lossy Functions from
Small-Exponent Hashing

Damiano Abram, Agni Datta, Archisman Dutta,
Lawrence Roy



eprint.iacr.org/2026/2367



Questions? I'm all ears.

BACKUP



Extra slides, in case you have questions.

Comparison

CONSTRUCTION	PUBLIC KEY	TRAPDOOR	OUTPUT	EVALUATION
ELF [Zha16]	$\Theta(n^2) G + \Theta(n) H$	–	$3n$ bits	$\Theta(n^2) \text{Op} + \Theta(n)$ hashes
TELF [Zha19]	$n^{3+o(1)} G$	$n^{3+o(1)} F$	$n^{3/2+o(1)} G$	$n^{3+o(1)} \text{Op}$
plain ELF	$n G$	–	$1 G$	$O(n) \text{Op}$
trapdoor ELF	$(2w + 2) G$	$1 E$	$1 G + 1 R$	$O(n) \text{Op}$

G, F, E, R, H : group, field, exponent, residue and hash units; Op : group operations; w : padded bit length of N .

Lower bound on the image size

THE REPEAT TEST

Sample k inputs; accept if they are distinct but two outputs agree.

- injective function: accepts with probability 0;
- image of size at most r : accepts with probability at least $1 - \exp(-\frac{k(k-1)}{2r}) - \frac{k(k-1)}{2m}$.

If no k -query algorithm has advantage above $1/\delta$ (with $\delta \geq 3$, $m \geq \delta k(k-1)$), then

$$r \geq \frac{k(k-1)}{2 \ln(1/(1 - 3/(2\delta)))} \geq \frac{(2\delta - 3)k(k-1)}{6} \geq \frac{\delta k^2}{12}.$$

For any ELF, $Q(n) \geq \delta(n)k(n)^2/24$, with $k(n)$ the evaluations a size- $t(n)$ observer can make.

Generic-group evidence

UPPER BOUND

A generic algorithm making T queries, given d powers, has advantage at most $\binom{T+d+1}{2}(\frac{1}{\gamma} + \frac{2}{\pi_q})$, where π_q is the smallest prime factor of the group order q .

Composite order is allowed.

MATCHING ATTACKS

Baby-step giant-step: $2\lceil\sqrt{\gamma}\rceil + 1$ queries, advantage $1 - \gamma/q$. A truncated grid with T queries gets at least $T^2/(8\gamma)$ if $\gamma \leq q/2$: the dependence on T is tight for one challenge.

PREPROCESSING

With polynomially many bits S of preprocessing on the encoding, a polynomial interval still suffices.

Listing the lossy image

A PREFIX DYNAMIC PROGRAM

Walk the bits of $m - 1$ from the top. Keep two sets of partial products:

- E : the prefix equal to that of $m - 1$;
- L : all smaller prefixes.

Each partial product is determined by a partial sum in $[0, n\gamma]$, so $|E| + |L| \leq r + 1$ throughout.

The image is listed exactly from the public key and r , in time $\text{poly}(r, \log m)$; for the plain ELF with $O(nr)$ group multiplications. A uniform element of the list is a uniform element of the image.

The full construction

SETUP AT IMAGE SIZE r

- sample (N, g) ; on failure, a fixed fallback key $hk_{\perp}$
- full r : g_0 and all $g_{j,k}$ uniform in $\mathbb{G}_N$ (any failed sample: $hk_{\perp}$)
- else $\bar{r} = \min\{r, 2^{\lambda}\}$, $\gamma = \lfloor (\bar{r} - 1)/(dw) \rfloor$;
 $g_0 = g^{s_0}$, $g_{j,k} = g^{s_{j,k}}$, s 's from $[\gamma]$ (all 1 if $\gamma = 0$)

HASH

- $h = g_0^{\rho'} \prod_{j,k} g_{j,k}^{x_{j,k}}$, ρ' the first
 $L_{\text{exp}} = \lceil \log_2 N \rceil + \lambda$ bits of the tape

GENERATE, ENCODE, DECODE

- σ from $[0..2^{L_{\text{exp}}} - 1]$, $K_0 = g_0^{\sigma}$,
 $K_{j,k} = g_{j,k}^{\sigma} \exp(2^k a_j)$
- $e = \text{DDLog}_N(K_0^{\rho'} \prod K_{j,k}^{x_{j,k}})$
- $e' = \text{DDLog}_N(h^{\sigma})$, so $e - e' = \sum_j a_j x_j$

INPUT PRIVACY

- at full r , if g_0 generates $\mathbb{G}_N$: $g_0^{\rho'}$ is within
 $2^{-\lambda-2}$ of uniform
- total distance at most $2^{-\lambda-1} + 2^{-\ell(\lambda)+4}$