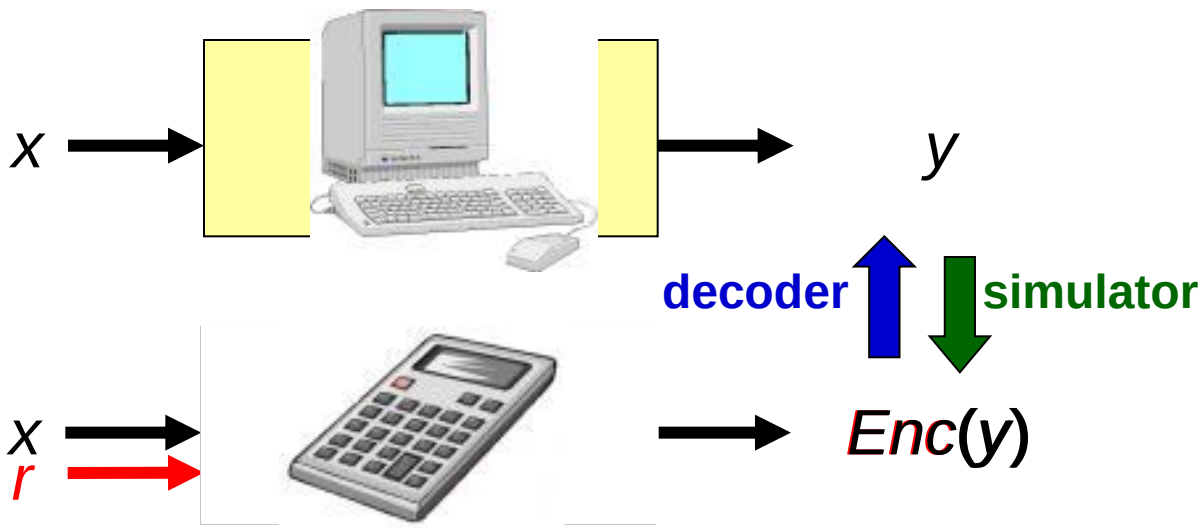
Randomly Encoding Functions: A New Cryptographic Paradaigm

Benny Applebaum

Electrical Engineering Department

Tel Aviv University

Encoding Functions



- *g* is a "randomized encoding" of *f* [IK00]
  - Nontrivial relaxation of computing *f*
- Hope:
  - *g* can be "simpler" than *f*
    (meaning of "simpler" determined by application)
  - *g* can be used as a substitute for *f*

Example: Addition [Kilian 88]

- Simple= Each output depends on 3 inputs
- $f(x_1,\ldots,x_n)=$ $x_1+x_2+\cdots+x_n \pmod 2$

simulator ↓ ↑ decoder

- $g(x, (r_1,\ldots,r_{n-1})) =$

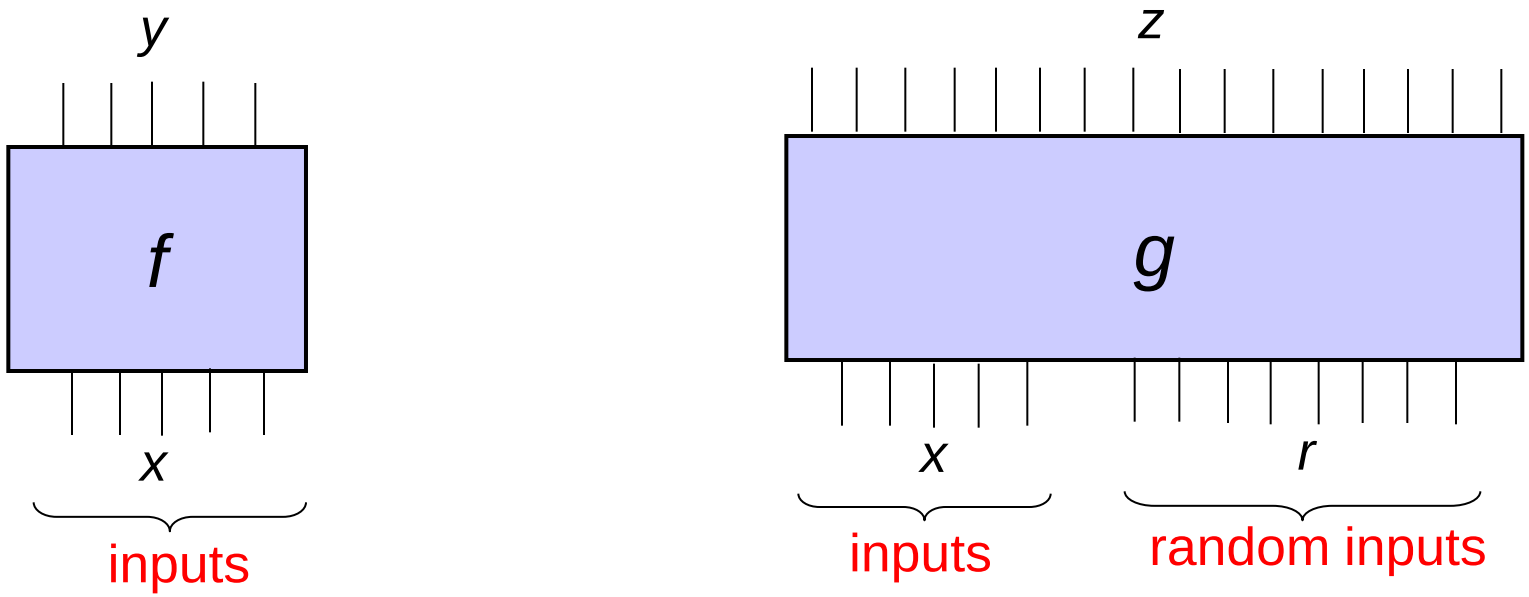
x_1+r_1 $-r_1+x_2+r_2$ $-r_2+x_2+r_3$ … $-r_{n-2}+x_{n-1}+r_{n-1}$ $-r_{n-1}+x_n$

- Generalizes to iterated group product

Plan

- Background
  - Definitions
  - Notions of simplicity
  - Which functions can be encoded?
- Sample of Applications
  - Delegation, Secure Computation
  - Parallel-Time Crypto
  - Key-dependent message security
- Out of Scope: How to construct RE's ?

Randomized Encoding - Syntax

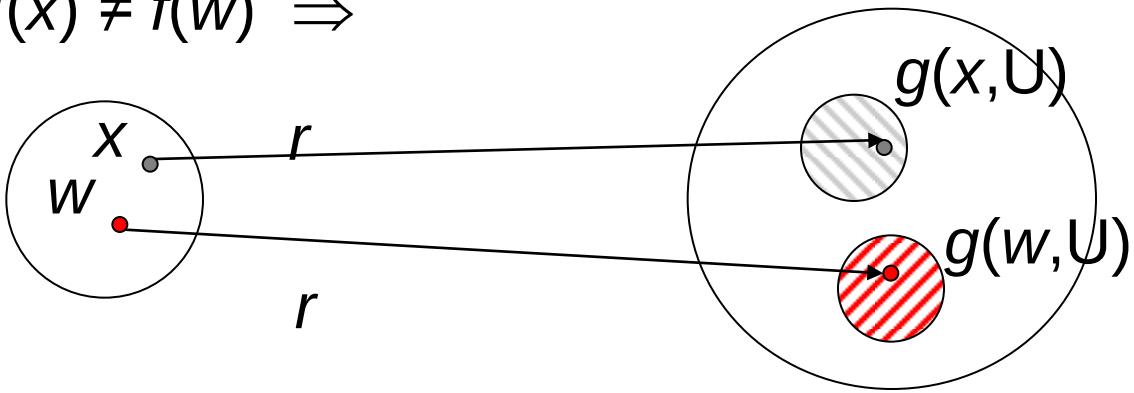


$f(x)$ is encoded by $g(x,r)$

Randomized Encoding - Semantics

- Correctness: $f(x)$ can be efficiently decoded from $g(x,r)$.

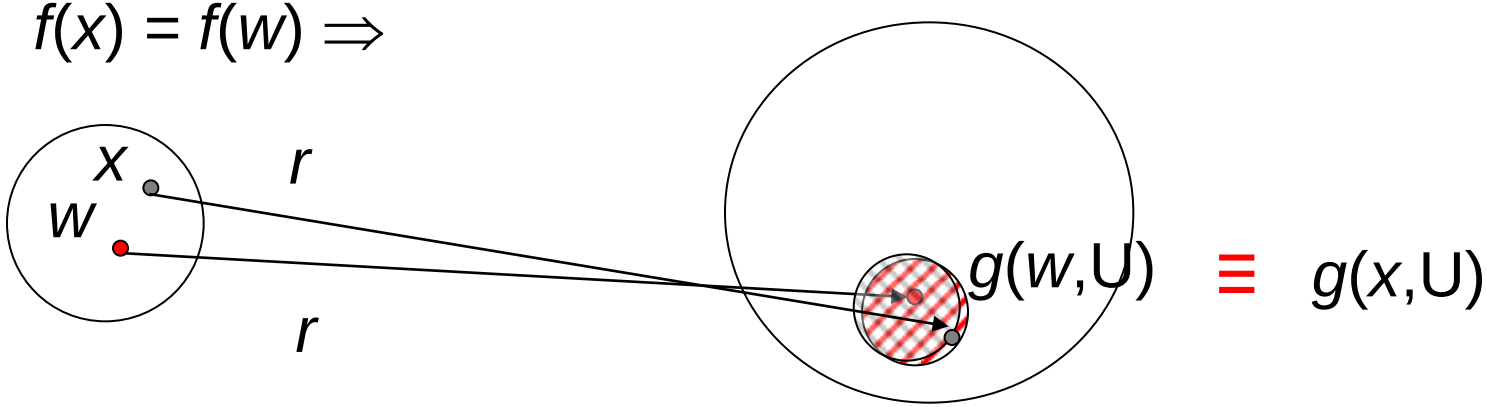
$f(x) \neq f(w) \Rightarrow$



- Privacy: $\exists$ efficient simulator Sim such that $\mathrm{Sim}(f(x)) \equiv g(x,U)$

  – $g(x,U)$ depends only on $f(x)$

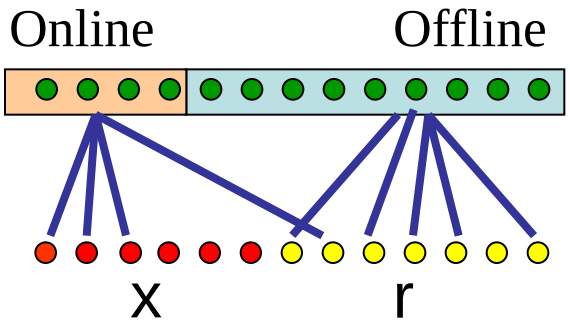
$f(x) = f(w) \Rightarrow$



Notions of Simplicity

Low online-complexity encoding

g(x,r)=(a(x,r),b(r))



Notions of Simplicity

Low online-complexity encoding

$g(x,r)=(a(x,r),b(r))$

Decomposable encoding

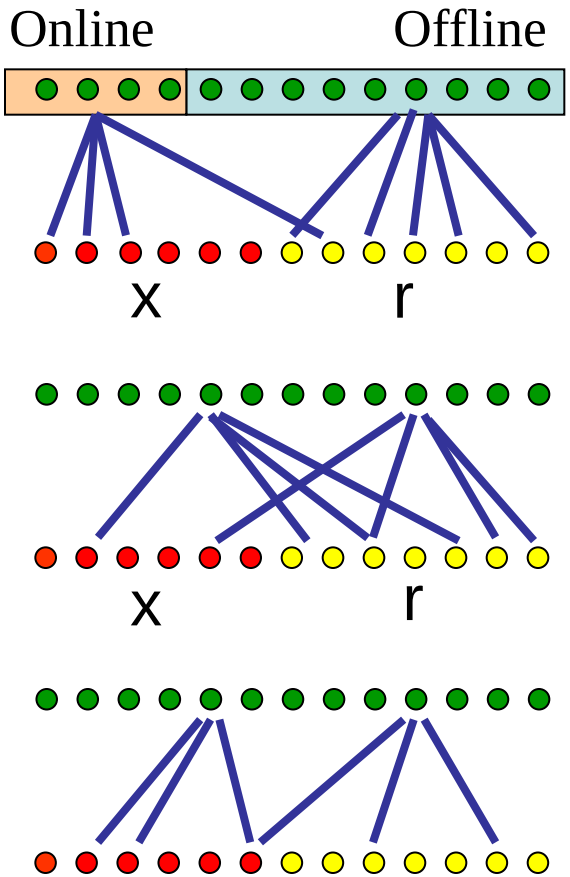
$g((x_1,\ldots,x_n),r)=(a_1(x_1,r),\ldots,a_n(x_n,r),b(r))$

NC^0 encoding

Output locality c

Low-degree encoding

Algebraic degree d over F



What can be encoded?

Thm [IK02,AIK04]: Every log-space computable function has a degree-3, locality-4, decomposable encoding.

- If we allow computational privacy then:

Thm [AIK05]: Every poly-time computable function has a degree-3, locality-4, decomposable, low-online encoding.

- Assuming log-space computable one-way function.
  (Implied by factoring, discrete-log, lattices…)

Applications

The Archetypical Story

Bob

Adversary

Alice

RE’s?

Functionality

Encoding the Functionality

Bob

Adversary

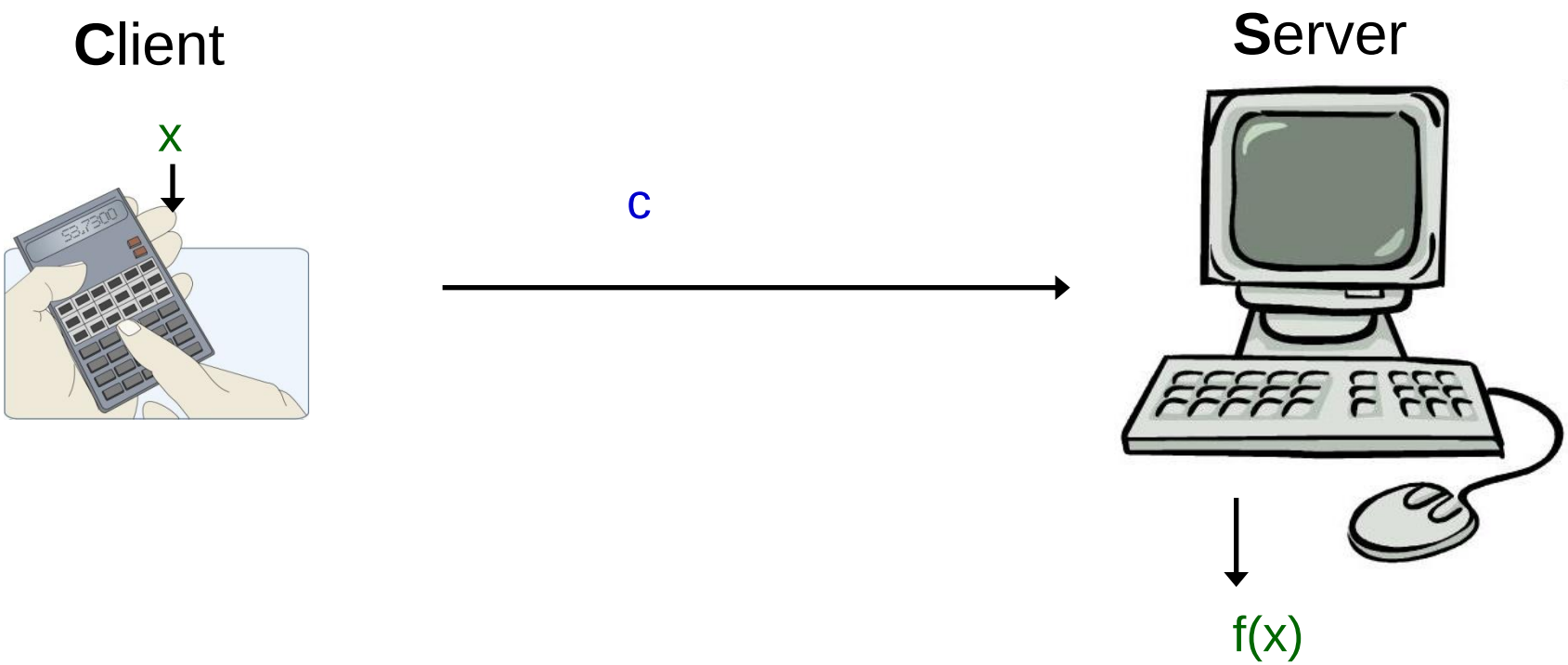
Alice

Functionality

Examples:

- Delegation
- Secure Computation

Basic Delegation



Goal: Server should learn f(x) without learning x

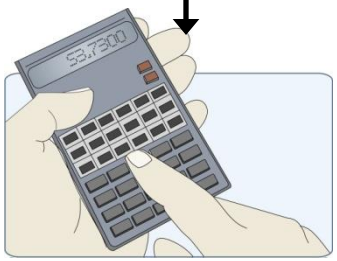
Non-Trivial if Client is too weak to compute f(x)

Sol: Send an RE of f

Client

Server

x



c=g(x;r)

Dec(c)=f(x)

- RE's gives the only known non-interactive solution
- If g(x;r) is "Simple"
  - Client can be Parallelized
  - Protocol has low online complexity

f(x)

⇩ ⇧

g(x;r)

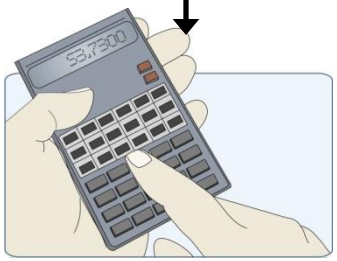
Sol: Send an RE of f

Client

Server

Offline Part a(r)

x



Online Part: b(x;r)

Dec(a,b)=f(x)

- RE's gives the only known non-interactive solution
- If g(x;r) is "Simple"
  - Client can be Parallelized
  - Protocol has low online complexity

f(x)

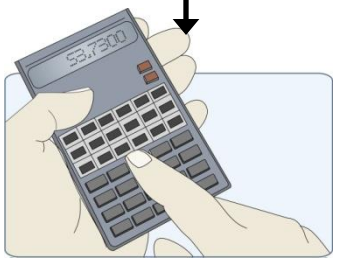
⇩ ⇧

g(x;r)

Sol: Send an RE of f

Client

x



Offline Part a(r)

Online Part: b(x;r)

Server

Dec(a,b)=f(x)

Useful Property:

- If Server cheats its output does not violate privacy
- i.e., erroneous output can be always simulated based on f(x)

f(x)

⇩ ⇧

g(x;r)

Verifiable Computation

Client

Server

x

y=f(x)

y

Input: x

Goal: compute y=f(x)

Verifiable Computation

Client

Server

$y=f(x)$

x →

← $y \neq f(x)$

Input: x

Goal: compute $y=f(x)$

Verifiable Computation: Client detects cheating

Correctable Computation: Client corrects "random" errors

RE+MAC $\Rightarrow$ VC [AIK10] implicit in [GGP10]

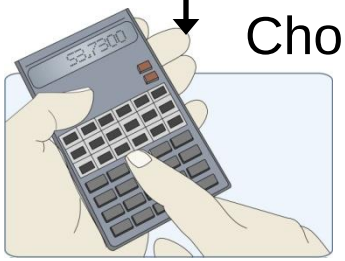
Client

Server

Encodes $MAC_k(f(x))$

x

Choose k



x, g(x,k;r)

a=f(x), b=Dec(g(x,k;r))

If $b=MAC_k(a)$ output a

- Idea: Ask **S** to compute f(x) + signature $\mathbf{MAC}_k(f(x))$ and verify consistency
- **S** should not know the key k !
- Define the mapping $(x,k) \rightarrow \mathbf{MAC}_k(f(x))$
- Let g(x,k;r) be an RE of this mapping

RE+MAC $\Rightarrow$ VC

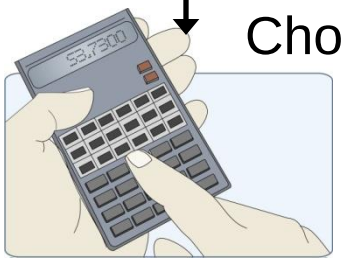
Client

Server

Encodes $MAC_k(f(x))$

x

Choose k



x, g(x,k;r)

a=f(x), b=Dec(g(x,k;r))

If $b=MAC_k(a)$ output a

- Correctness: follows from correctness of RE
- Soundness: If **S** is able to cheat then can forge the **MAC**

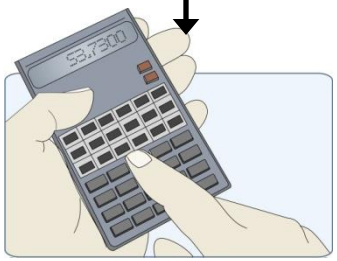
RE+MAC $\Rightarrow$ VC

Client

Server

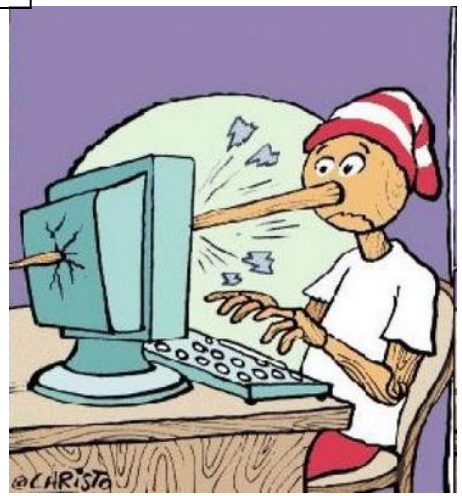
Encodes $MAC_k(f(x))$

x



$x, g(x,k;r)$ →

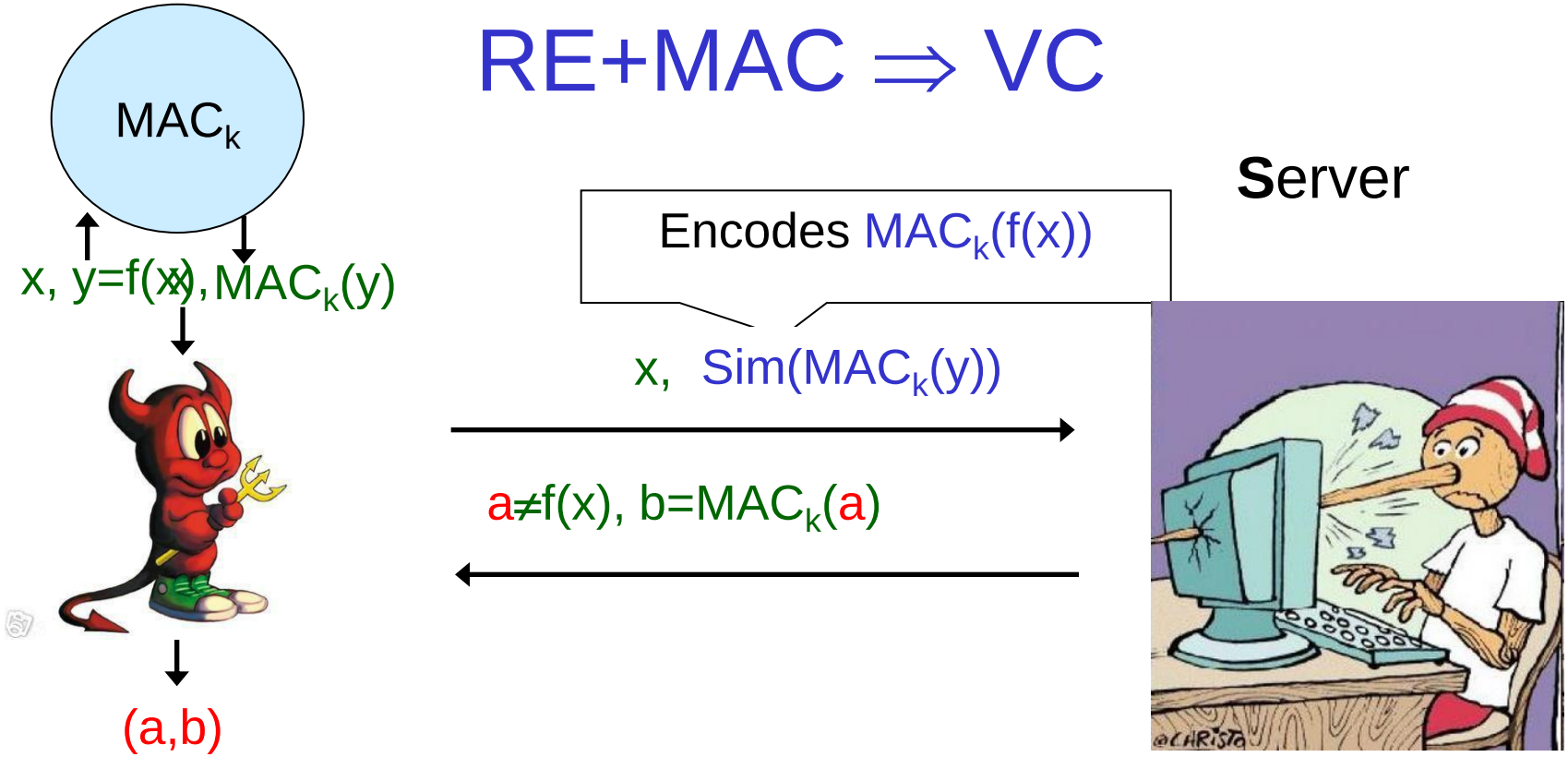
← $a \neq f(x), b=MAC_k(a)$



If $b=MAC_k(a)$ output a

- Correctness: follows from correctness of RE
- Soundness: If **S** is able to cheat then can forge the **MAC**

RE+MAC ⇒ VC



- Correctness: follows from correctness of RE
- Soundness: If **S** is able to cheat then can forge the **MAC**

RE+MAC ⇒ VC

Client

Server

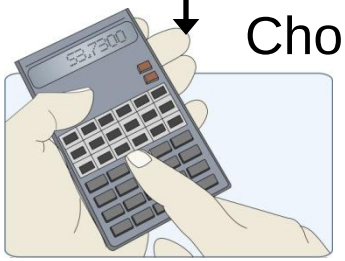
Offline g(x,k;r)

x

Choose k

x, Online g(x,k;r)

a=f(x), b=Dec(g(x,k;r))



If b=MAC_k(a) output a

- Corollary: Every poly-time function has VC with low online complexity
- Alternative approaches: CS Proofs [Mic94], FHE [CKV10]
- Can add secrecy/correctability by combining RE's with one-time pads
- Corollary: NC^0 **Program correctors** for log-space (strengthen [GGHKR07])

REs and Secure Computation

Bob

x_B

x_A

Alice

$\Leftrightarrow$

Functionality
$f(x_A, y_B)$

Encoding
$g(x_A, x_B; r)$

Example: Decomposable Encoding

Bob

$g_A(x_A;r)$

$\mathbf{x_B}$

$\mathbf{r}$

$\mathbf{x_A}$

Alice

x_i

$a_i(x_i;r)$

OT

$a_i(0;r)$

$a_i(1;r)$

$g(x_A,x_B;r)$

Decoder

$f(x_A,x_B)$

Functionality $\Leftrightarrow$ Decomposable encoding

$\mathbf{f\ (x_A,y_B)}$

$g((x_1,\ldots,x_n),r)=(a_1(x_1,r),\ldots,a_n(x_n,r),b(r))$

REs and Secure Computation

Thm.[IK00]: Securely computing f reduces to securely computing g

- decomposable g reduces to OT
- low degree g has constant round protocol [BGW88,CCD88,CDM00,…]

Paradigm for efficient secure computation :

- Encode **complex** functions by **simple** ones
- Derive a protocol for **complex** from a protocol for **simple**

Examples:

- [IK00] Information-theoretic O(1)-round protocols for log-space
- [IKOPS11] Non-interactive computational secure computation for P-time
- Simple proofs for classics: Completeness of OT [Kil 88]

  O(1) round protocols for P-time [BMR90]

Encoding the Honest Parties

Bob

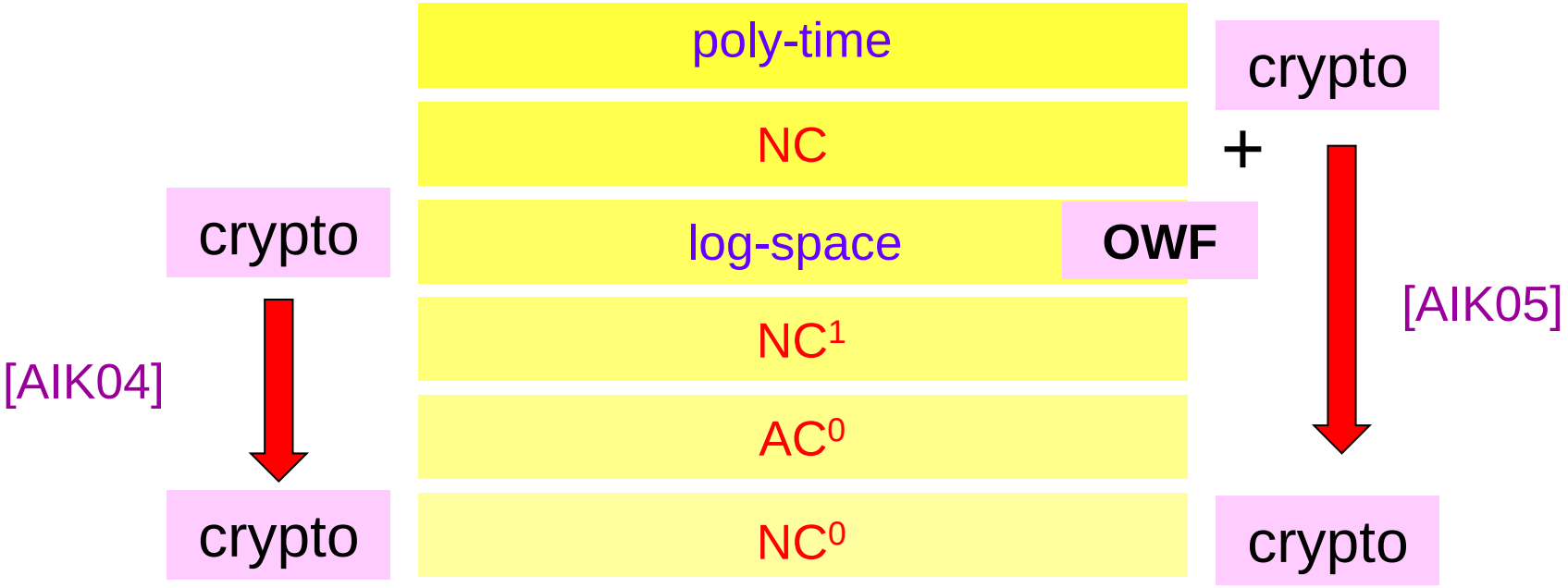
Adversary

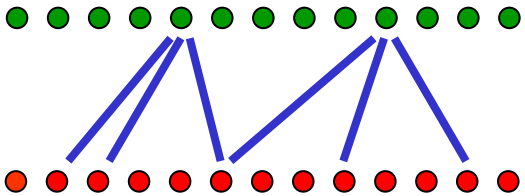
Alice

Functionality

Parallel Cryptography

How low can we get?





Too weak for crypto?

Encoding an Encryption Function

sk

$C = E(pk, m)$

pk

m

$D(sk, C)$

Encoding an Encryption Function

sk

pk

m

D(sk,C)

Encoding an Encryption Function

sk

$\downarrow$

D(sk,C)

C'=Encoding(E (pk, m))

$\longleftarrow$

m

pk

Encoding an Encryption Function

sk

$C' = \text{Encoding}(E(pk, m))$

pk

m

$C = \text{Decode}(C')$

$D(sk, C)$

Encoding an Encryption Function

sk

$\downarrow$

C=Decode(C')

D(sk,C)

C'=Encoding(E (pk, m))

$\longleftarrow$

m

pk

Claim: Breaking the new scheme

$\Rightarrow$ Breaking the original

Encoding the Encryption Function

C'=Encoding(E (pk, m))

pk

C=E (pk, m)

pk

Claim: Breaking the new scheme

$\Rightarrow$ Breaking the original

Encoding the Encryption Function

$C' = \text{Encoding}(E(pk, m))$ ←

pk

C' ← Sim ← $C = E(pk, m)$

pk

Claim: Breaking the new scheme

$\Rightarrow$ Breaking the original

Crypto with Low Complexity from RE

- Thm. [AIK04]: RE preserves crypto hardness of most primitives
  - E.g., OWF, OWP, PRG, Sym-Enc, PK-Enc, Sign, MAC, Hash, Com, ZK
  - Also works for information-theoretic primitives (ε-biased gens, extractors,…)
  - Different primitives require different variants of randomized encoding

- Paradigm for crypto with low complexity:
  - Encode functions in complexity class **HIGH** by functions in **LOW**
  - Show that a primitive **P** can be implemented in **HIGH**
  - Conclude that **P** can be implemented in **LOW**

Encoding the Adversary

Bob

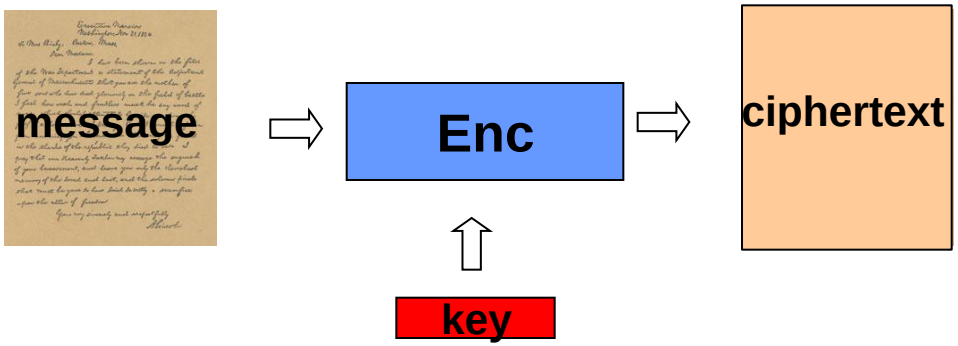
Adversary

Alice

Functionality

Key-Dependent Message Security

Standard Security: $\forall$ message, **random** key $\Rightarrow$ ciphertext **"hides"** message



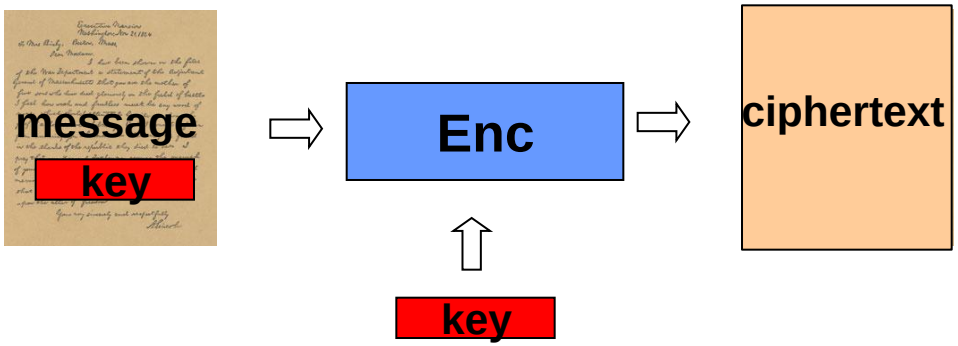
Key-Dependent Message Security

Standard Security: $\forall$ message, **random** key $\Rightarrow$ ciphertext **"hides"** message

What if the message **depends** on the key ? [GM82]

-E.g., $E_{key}(key)$ or $E_{key}(f(key))$

F-KDM Security [CL01, BRS02] : Adversary can ask for $E_k(f(k))$ for all $f \in F$



KDM Amplification: $G \Rightarrow_{KDM} F$

Goal: Given G-KDM encryption construct F-KDM encryption

Want:

- Large Gap (minimize G, maximize F)
- Generality: works for every scheme

Previous works [BGK09, BHHI10]:

- Large gap (G=affine, F= Fixed poly-time computable family)
- Need additional (seemingly stronger) properties than KDM
  - entropic-KDM, simulatable KDM
  - Can't amplify the Learning-Parity-with-Noise based scheme

Coming up: Generic amplification with large gap **[A11]**

If G encodes f then $G \Rightarrow_{KDM} f$

G "encodes" f if: $g(x;r)$ encodes $f(x)$,

$g_r(x)$ is in G for every fixed r

If G encodes f then $G \Rightarrow_{KDM} f$

sk

$\downarrow$

D(sk,C)

C=E (pk, m)

$\longleftarrow$

m

pk

G-KDM Adversary

G "encodes" f if: g(x;r) encodes f(x),

$g_r(x)$ is in G for every fixed r

If G encodes f then $G \Rightarrow_{KDM} f$

sk

pk

m

D(sk,C)

G-KDM Adversary

G "encodes" f if: g(x;r) encodes f(x),

$g_r(x)$ is in G for every fixed r

If G encodes f then $G \Rightarrow_{KDM} f$

sk

$C = E(pk, g_r(sk))$

pk

m

$D(sk, C)$

G-KDM Adversary

G "encodes" f if: $g(x;r)$ encodes $f(x)$,

$g_r(x)$ is in G for every fixed r

If G encodes f then $G \Rightarrow_{KDM} f$

sk

$D(sk, C)$

$C = E(pk, g_r(sk))$

pk

m

G-KDM Adversary

G "encodes" f if: $g(x;r)$ encodes $f(x)$,

$g_r(x)$ is in G for every fixed r

If G encodes f then $G \Rightarrow_{KDM} f$

sk

$\downarrow$

D(sk,C)

C=E (pk, m)

$\longleftarrow$

m

pk

G "encodes" f if: g(x;r) encodes f(x),

$g_r(x)$ is in G for every fixed r

If G encodes f then $G \Rightarrow_{KDM} f$

sk

$\downarrow$

D(sk,C)

pk

m

G "encodes" f if: g(x;r) encodes f(x),

$g_r(x)$ is in G for every fixed r

If G encodes f then $G \Rightarrow_{KDM} f$

sk

$\downarrow$

D(sk,C)

C=E'(pk,m)=

$\longleftarrow$

E (pk, **Sim(**m))

m

pk

G "encodes" f if: g(x;r) encodes f(x),

$g_r(x)$ is in G for every fixed r

If G encodes f then $G \Rightarrow_{KDM} f$

sk

$\downarrow$

D(sk,C)

m=Decode(m')

C=E'(pk,m)=

$\longleftarrow$

E (pk, **Sim(**m))

m

pk

G "encodes" f if: g(x;r) encodes f(x),

$g_r(x)$ is in G for every fixed r

If G encodes f then $G \Rightarrow_{KDM} f$

sk

$C = E'(pk, m) =$

$E(pk, \mathbf{Sim}(m))$

m

pk

$m' = D(sk, C)$

$m = Decode(m')$

G "encodes" f if: $g(x;r)$ encodes $f(x)$,

$g_r(x)$ is in G for every fixed r

Security

f-KDM for E'

$E'(pk, f(sk))$

$= E(pk, Sim(f(sk)))$

$\approx E(pk, g(sk,r))$

pk

G-KDM for E

$E(pk, g_r(sk))$

pk

Note: Easily Generalizes to function families

Corollary: projections $\Rightarrow_{\mathbf{KDM}}$ fixed poly-size circuits

Summary: Applications

- Randomized encoding are useful
  - Delegation
  - Secure computation
  - Cryptography with low complexity
  - KDM security
  - Computing over encrypted data [SYY99,CCKM01]
  - Complete problems for zero-knowledge complexity classes [DGRV11]
  - More?

Summary: Constructions

- Information-theoretic construction for Log-space
  - Extend to P? to NC^2?
- Computational constructions for P
  - Based on log-space computable one-way functions
  - Any OWF?
- New construction for arithmetic circuits [AIK11]
- Many Open questions
  - Optimal parameters? (e.g. locality)
  - Manipulating RE's?

Thank You !