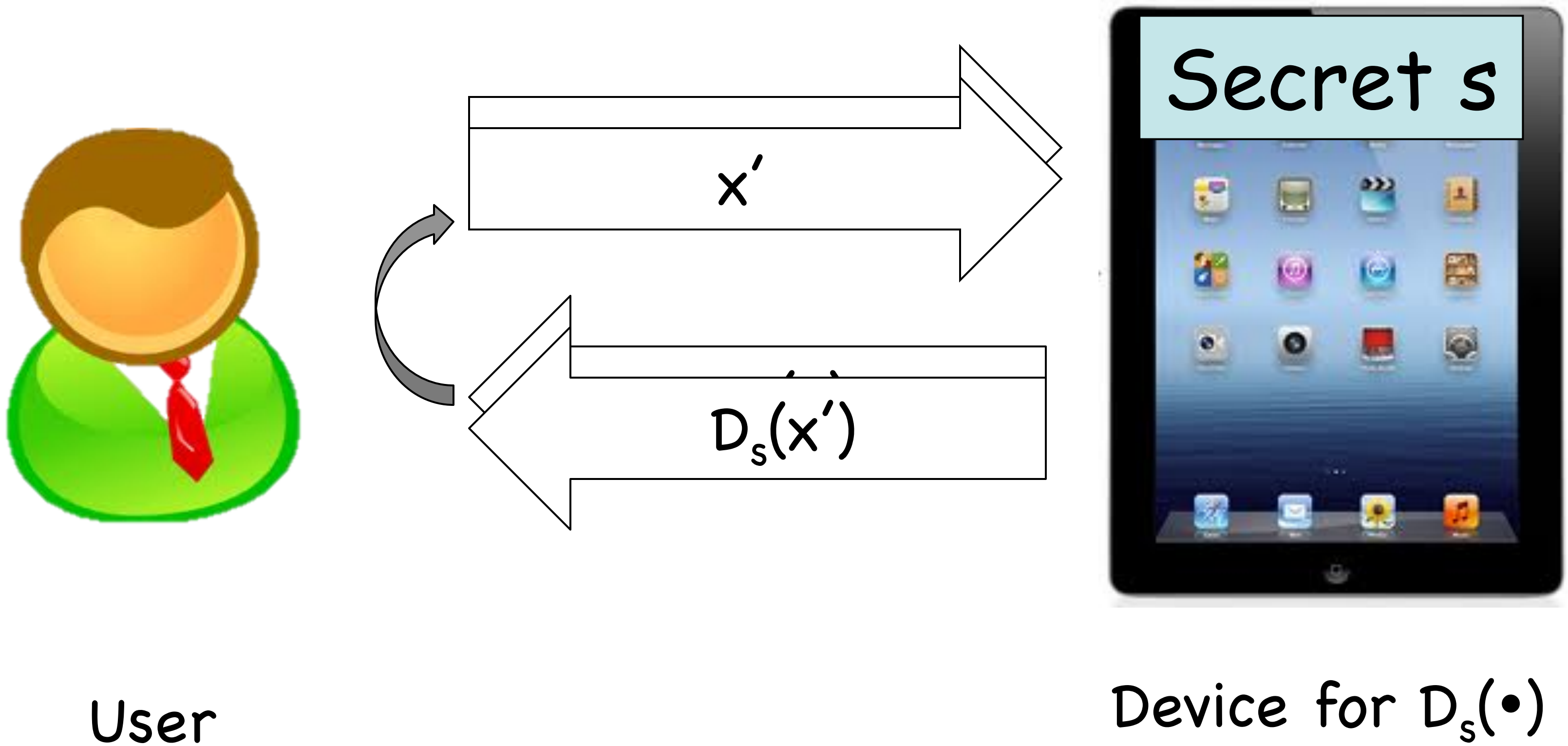
Tamper and Leakage Resilience in the Split State Model

Feng-Hao Liu

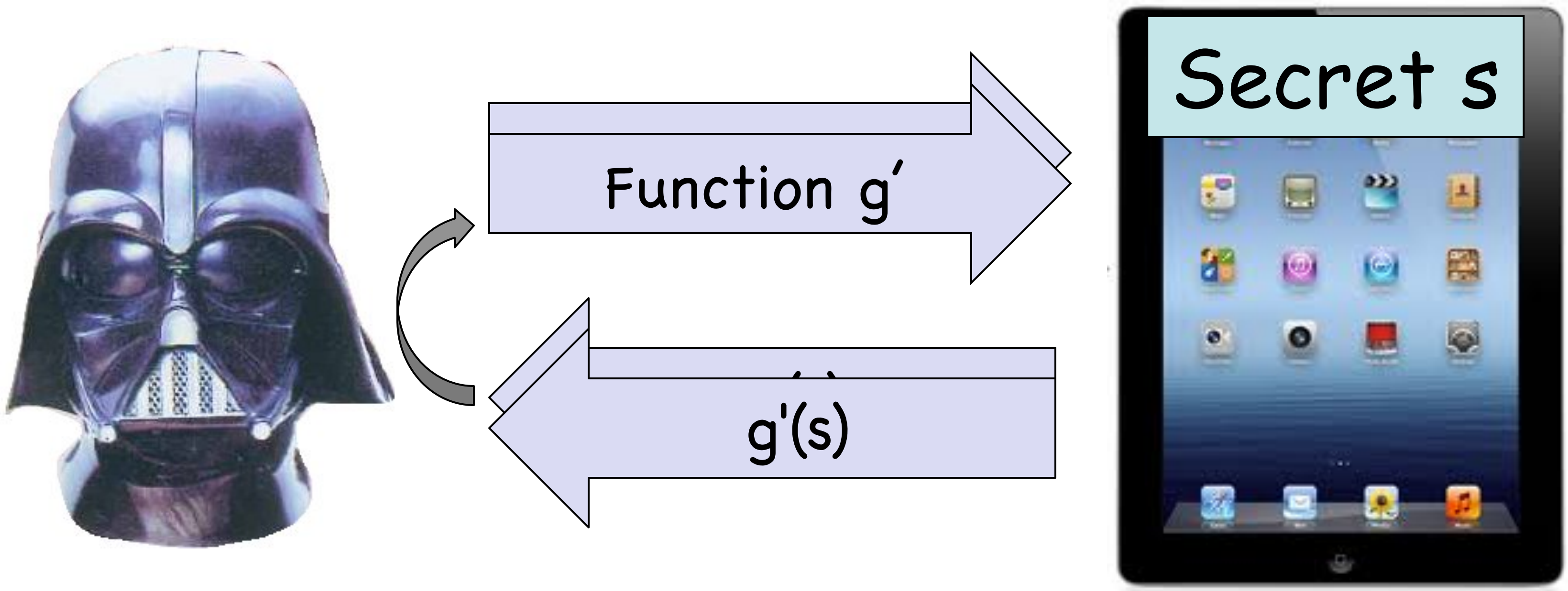
Anna Lysyanskaya

Brown University

I/O Access to a Device



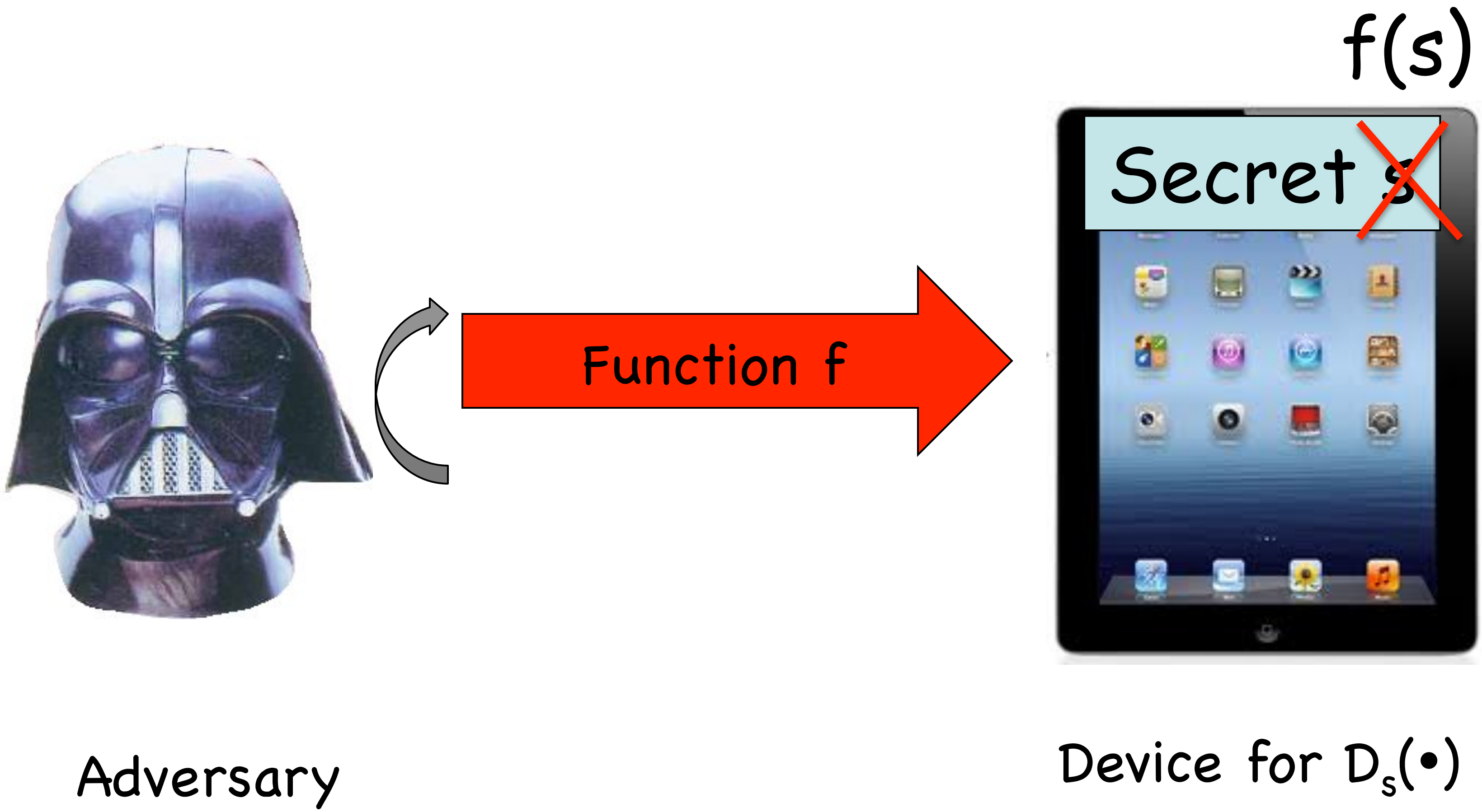
Leakage/Side Channel Attack [Kocher96,HSHCP+08…]



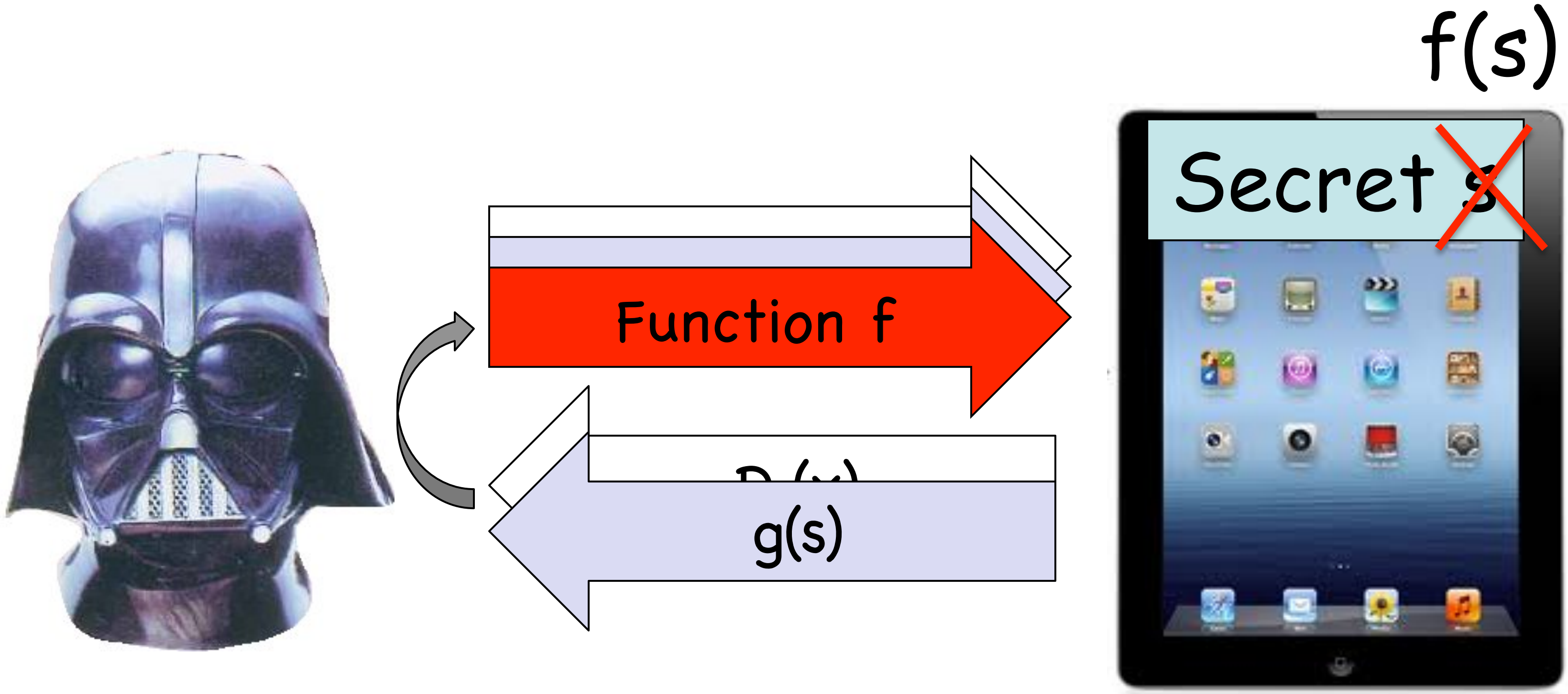
Adversary

Device for $D_s(\bullet)$

Tampering Attack [BS97,AARR02...]



Continuous L+T Attack [LL10,KKS11]



Adversary

Device for $D_s(\bullet)$

Remark: in this model, computations (updates) happen between the attacks, unlike the model of [MR04]

Previous Work

- Tampering only [GLMMR04,DPW10,CKM11]
- Leakage only [AGV09,NS09,KV09,
DHLW10,BKKV10,DLWW11...]
- Combined attack:
  - Negative results: [LL10] how to leak from the future if have no randomness, even in very restricted attack models
  - Positive results: [KKS11] encryption and signatures if have fresh randomness on update

Where do you get fresh randomness while under attack??

Our Goal

- An architecture that can tolerate leakage and tampering attacks at the same time…
  - without assuming an on-device source of randomness
  - under a reasonable restriction on the adversary's leakage and tampering power

Our Main Result

- A compiler: given D, produces SecureD

Compiler

$D_S(\bullet)$

- Identical I/O behavior
- SecureD is leakage and tamper resilient in the split state model with a CRS

To be defined

$\text{SecureD}_M(\bullet)$

The Split State Model [DPW10,DLWW11,HL11...]

The two parts are attacked separately

Split State Model: Leakage

Functions g_1, g_2

$g_1(M_1)$, $g_2(M_2)$

Adversary

Device for $D_{M1,M2}(\bullet)$

Split State Model: Tampering

$f_1(M_1)$

Functions f_1, f_2

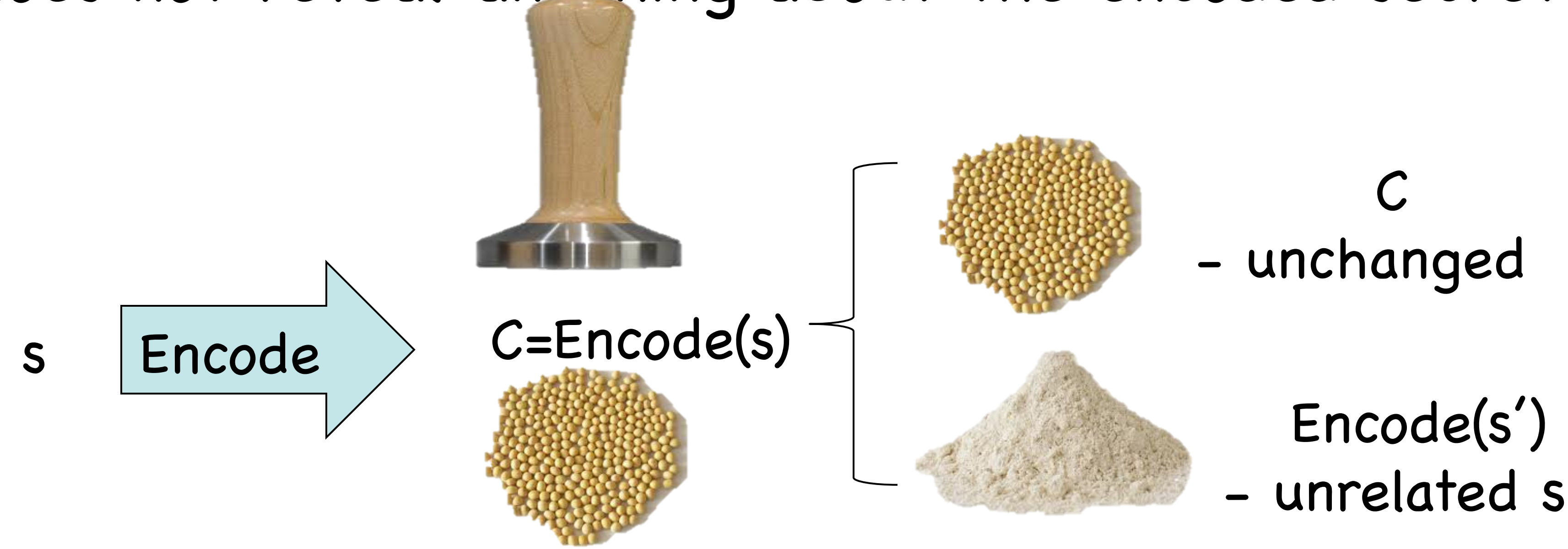
$f_2(M_2)$

Adversary

Device for $D_{M1,M2}(\bullet)$

Towards Tamper-Resilience: Non-Malleable Code [DPW10]

- Encode s into C = (M_1,M_2) s.t. tampering is useless!
- Non-malleable code [DPW10]: “mauling” the code does not reveal anything about the encoded secret



Non-Malleable Code

- Formally: for all f in F, all s, s',

$$\mathrm{Tamper}(f,s) \approx \mathrm{Tamper}(f,s')$$

ppt

$$\mathrm{Tamper}(f,s) = \begin{cases} \text{"same" if } f(c) = c, \text{ where } c \leftarrow \mathrm{Encode}(s) \\ \mathrm{Decode}(f(c)) \text{ otherwise} \end{cases}$$

CRS

CRS

CRS

Impossible in general, but

- [DPW10] construct them in SS RO model, unbounded functions
- We construct them in SS CRS model, poly-sized functions

NM Code Protects from Tampering Attack [DPW10]

Adversary's view in real attack:

Function f

C = ~~Encode~~(s)

Decode(f(C))

Adversary's view in a simulation:

Function f

C'=~~Enc~~(000)

Decode(f(C'))

NM Code in the SS Model: Our Construction

Encode(s) = | M_1 = sk | $M_2 = (pk, C=Enc_{pk}(s), \pi)$

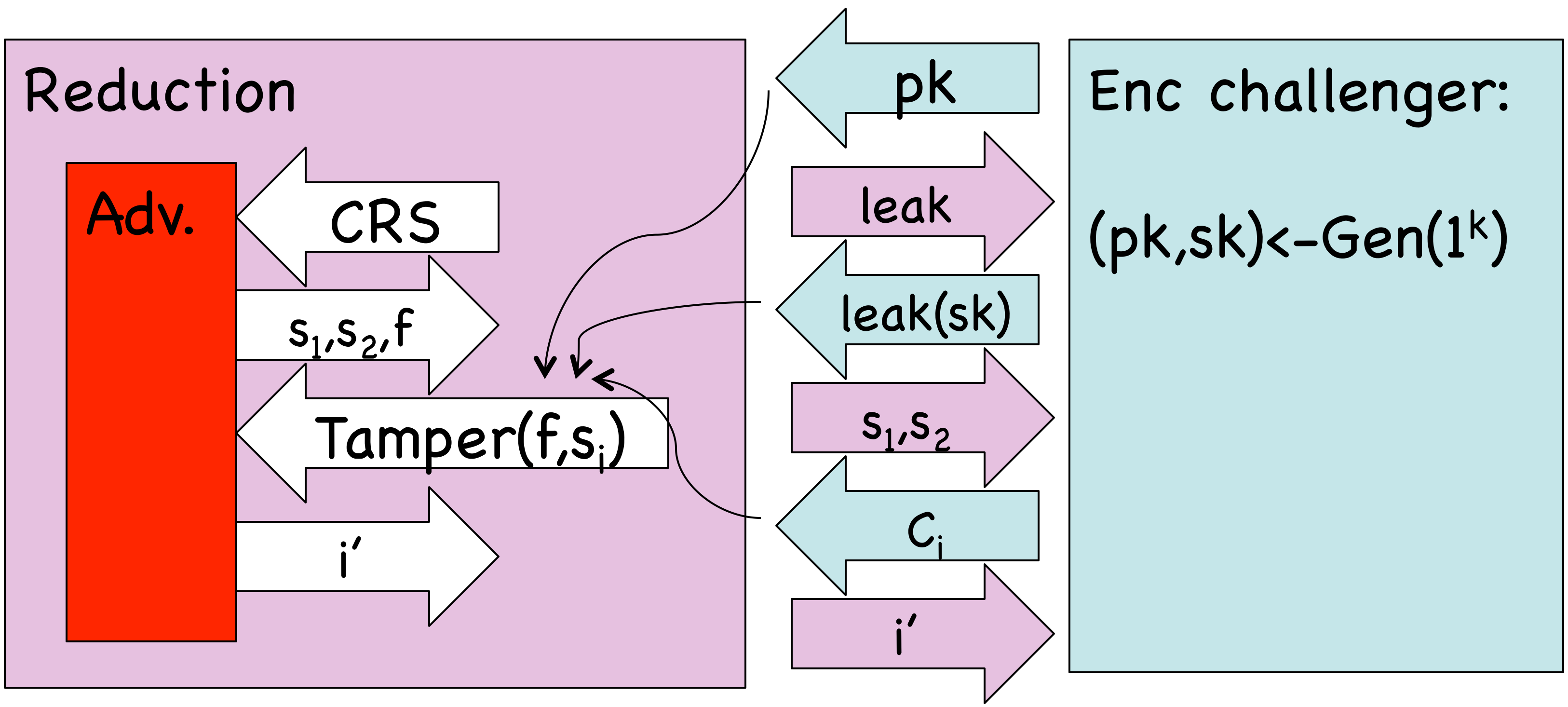
- the encryption scheme is leakage-resilient
- unique pk for each sk, unique sk for each pk (*)
- π is a non-malleable (robust) NIZK PoK of sk and the decryption of C

Robust NIZK PoK [DDOPS01]: even with access to a ZK simulator, Adv can only produce proofs whose witnesses can be extracted (similar to UC NIZK)

Proof of Security of Our NM Code

Suppose this is not a NM code. Then there exist s_1, s_2, $f=(f_1,f_2)$ such that Tamper(f,s_1) can be distinguished from Tamper(f,s_2).

Use that to break encryption:



Proof [illegible]

$$\text{Tamper}(f,s) = \begin{cases} \text{“same” if } f(c) = c, \text{ where } c \leftarrow \text{Encode}(s) \\ \text{Decode}(f(c)) \text{ otherwise} \end{cases}$$

Suppose this [illegible])
such that Tar[illegible]

Use th[illegible] h: (pk, $C_1=\text{Enc}_{pk}(s_1)$) from (pk, $C_2=\text{Enc}_{pk}(s_2)$):

(Hopefully) don't need

Reduction computes Tamper(f,s_i) as f[illegible]

M_1 = sk, M_2 = (pk,C_i,π)

Use ZK simulator

recall $f = (f_1,f_2)$

$M_2' = (pk',C',\pi') = f_2(M_2)$;

if $M_2' \neq M_2$ then:

if π′ invalid, output NULL

else extract (s′,sk′) from π′

if sk′ = $f_1(M_1)$ output s′, else NULL

Robust NIZK Pok

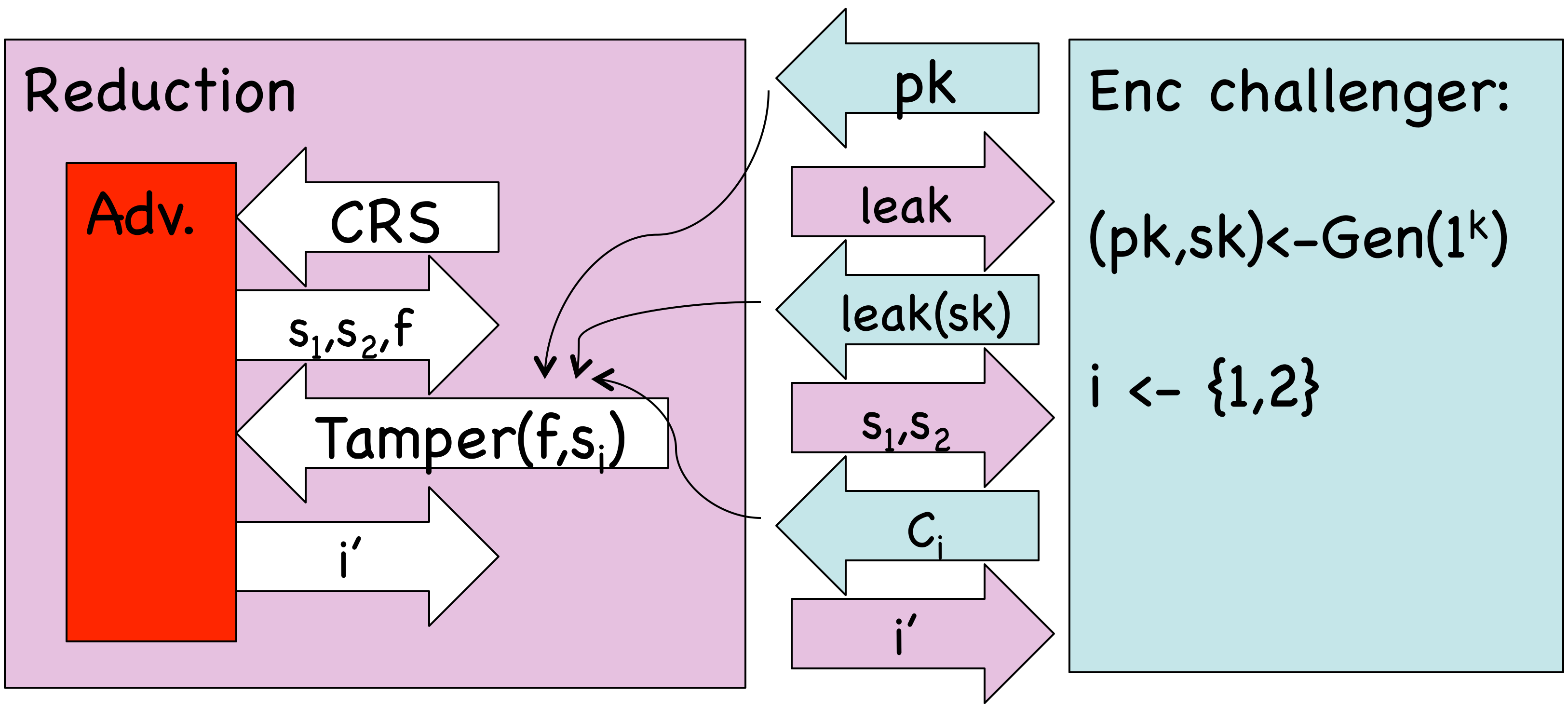
else if $M_2'=M_2$: if $f_1(M_1)$=sk output [illegible] ULL

Use leakage query here

Proof of Security of Our NM Code

Suppose this is not a NM code. Then there exist s_1, s_2, $f=(f_1,f_2)$ such that Tamper(f,s_1) can be distinguished from Tamper(f,s_2).

Use that to break encryption:

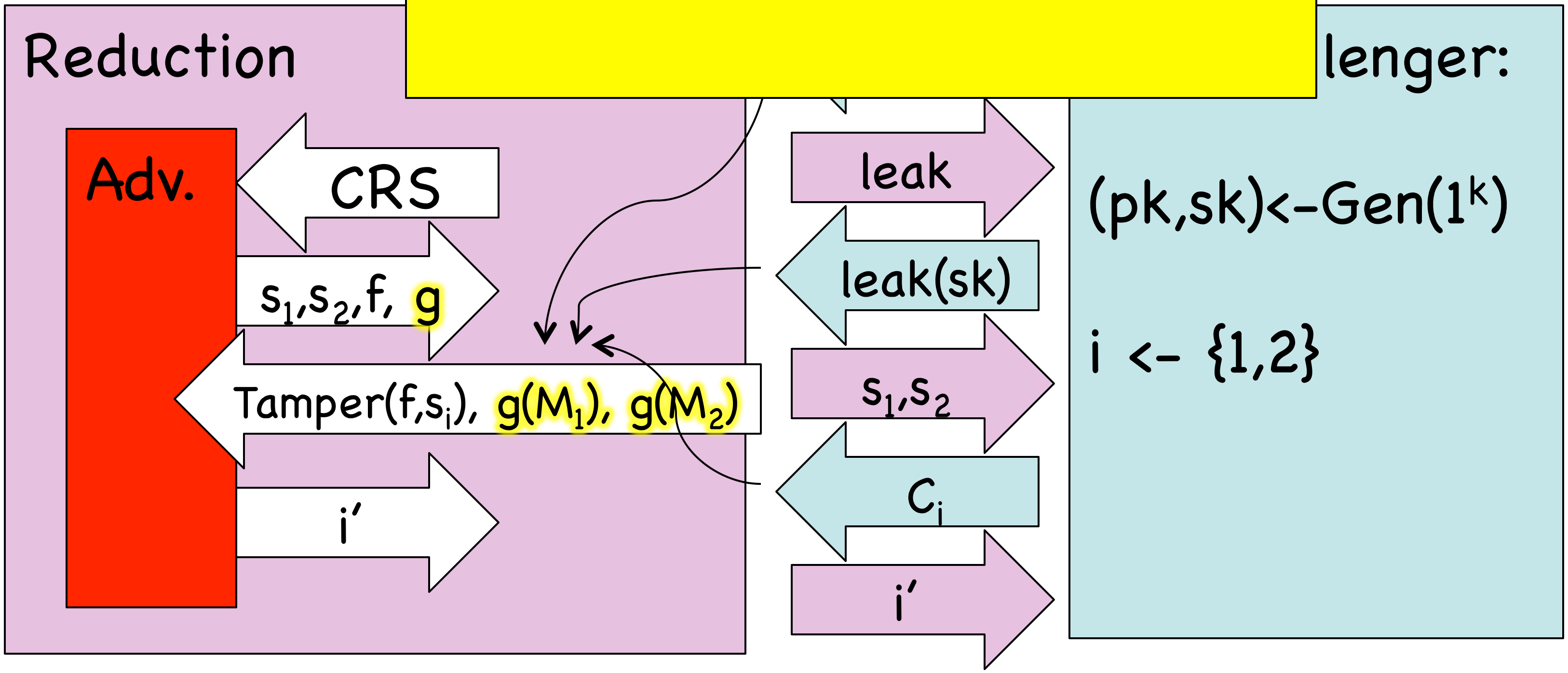


Proof of Security of Our NM Code

Suppose this is not a NM code. Then there exist s_1, s_2, $f=(f_1,f_2)$ such that Tamper(f,s_1) can be distinguished from Tamper(f,s_2).

Use that to break e

Bonus!! Our NM code is also leakage-resilient in SS model!



LR NM Codes => L+T Resilience

- A compiler: given D, produces SecureD

Compiler

$D_s(\bullet)$

- Identical I/O behavior
- SecureG is leakage and tamper resilient in the split state model with a CRS

Defined

$\text{SecureD}_{M1,M2}(\bullet)$

Our L+T Resilient Compiler: Construction 1 (Randomized)

$D_s(\bullet)$

Compiler

(M_1,M_2) = Encode(s) using the LR NM code

$\text{SecureD}_{M1,M2}(x)$:

s = Decode(M_1,M_2)

refresh: (M_1,M_2) <- Encode(s)

output $D_s(x)$

Here use randomness

Our L+T Resilient Compiler: Construction 2 (Deterministic)

$D_s(\bullet)$

Compiler

(M_1,M_2) = Encode(s,seed) using the LR NM code

Secure$D_{M1,M2}(x)$:
    (s,seed) = Decode(M_1,M_2)
    rand, seed′ = PRG(seed)
    refresh: (M_1,M_2)= Encode(s,seed′) using rand as coins
    output $D_s(x)$

CONCLUSION

Traded off perfect randomness for SS model:

- got L+T resilience for EVERY functionality
- after-the-fact leakage and tampering resilience (solved open problems of [HL11])
- achieved simulation based security

Of independent interest: new NM code (solved open problems of [DPW10])