Detection of Algebraic Manipulation

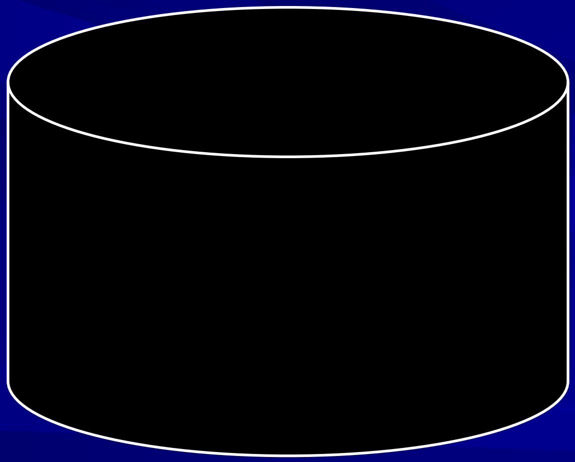
with Applications to Robust Secret Sharing and Fuzzy Extractors

Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padro, Daniel Wichs

Abstract Storage Device $\Sigma(G)$

- **<u>Properties of $\Sigma(G)$:</u>**

1. $\Sigma(G)$ provides **privacy**.
2. $\Sigma(G)$ allows **algebraic manipulation**.



Storage $\Sigma(G)$

$\Delta \in G$

Abstract Storage Device $\Sigma(G)$

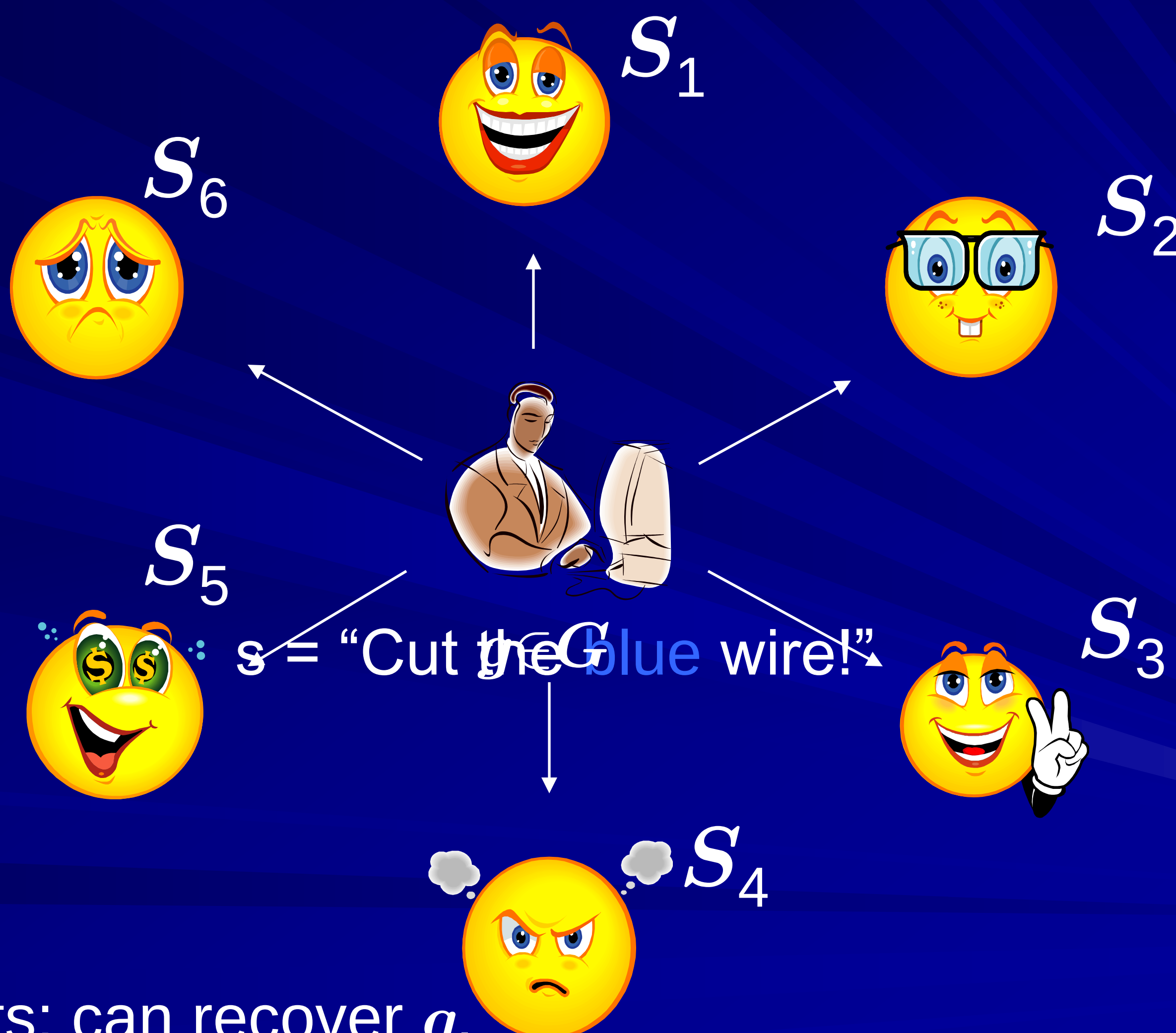
- Q: Why study devices with these properties?

- A: They appear implicitly in crypto applications:
  - Secret Sharing
  - Fuzzy Extractors

- Problem: Because of algebraic manipulation, the above primitives are vulnerable to certain **active** adversarial attacks.

- Task: A general method that helps us add "robustness" to secret sharing and fuzzy extractors.

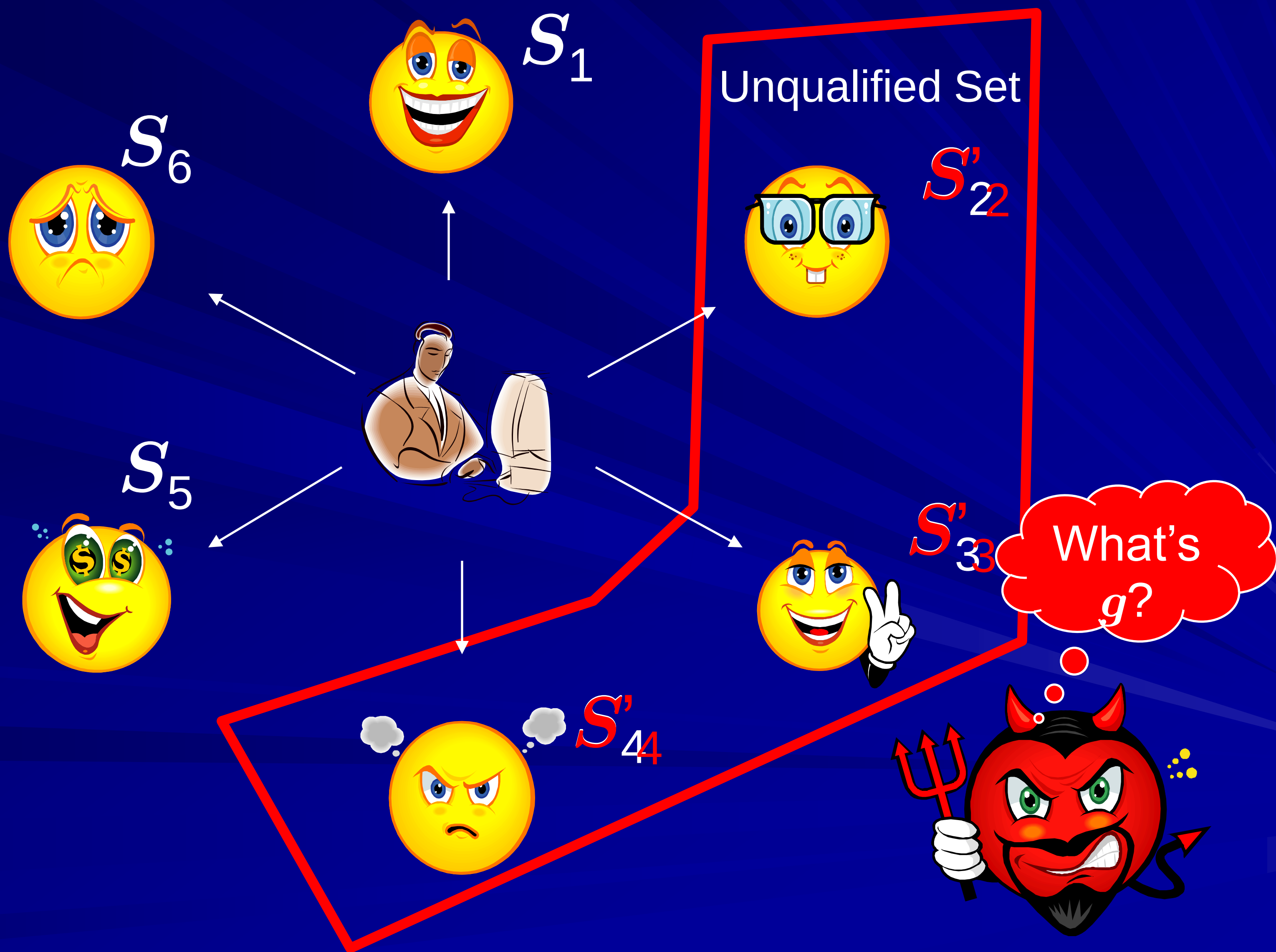
Application: Secret Sharing



Qualified sets: can recover g.

Unqualified sets: learn nothing about g.

Application: Secret Sharing



Application: Secret Sharing

Robust Secret Sharing: An adversary who corrupts an *unqualified* set of players cannot cause the recovery of $s' \neq s$.

Linear Secret Sharing

Assume Secret Sharing is Linear (i.e. [Sha79,KW93,…])

$$g' = \mathrm{Rec}(S_1, S'_2, S'_3, S'_4, S_5, S_6)$$

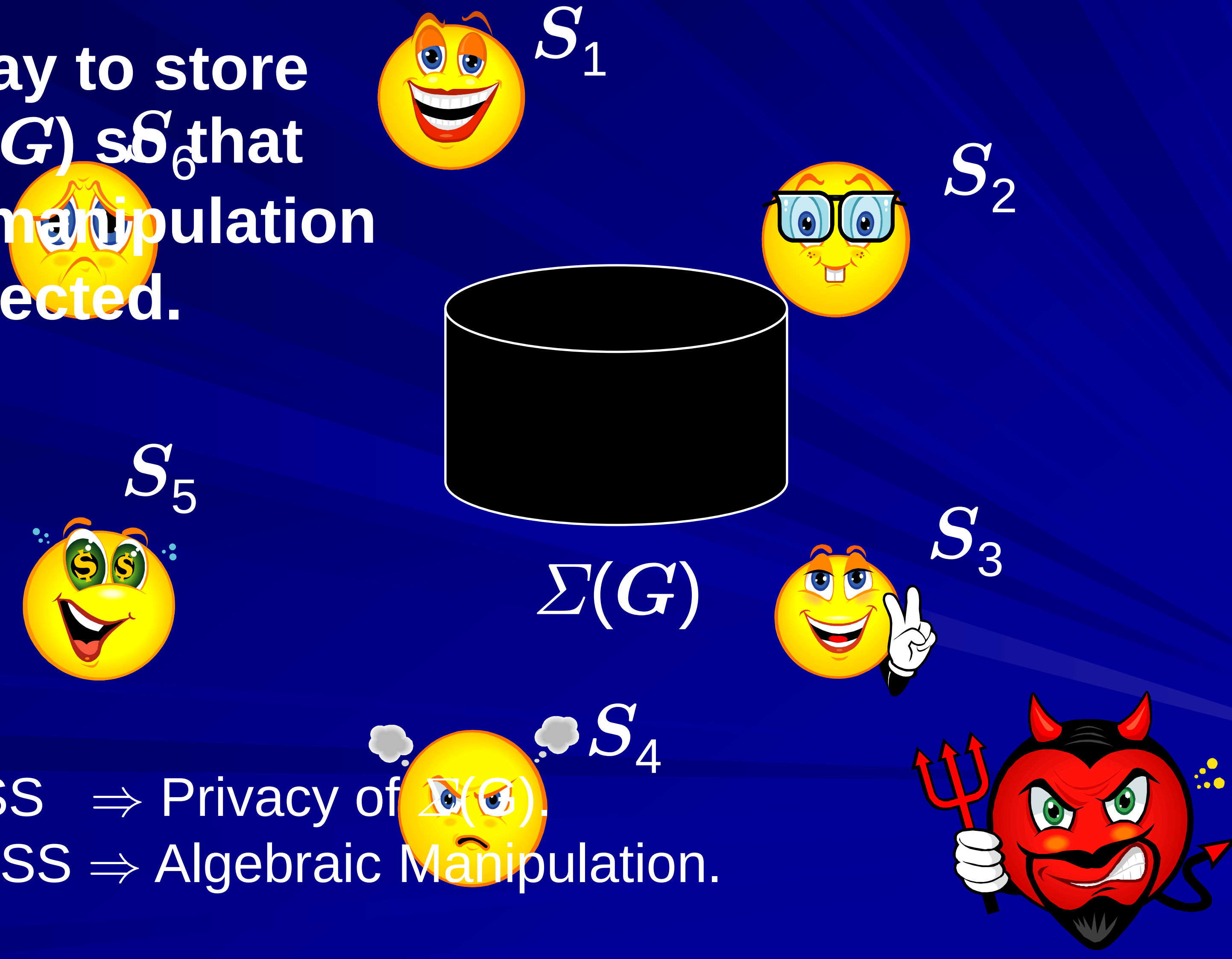
$$= \mathrm{Rec}(S_1, S_2, S_3, S_4, S_5, S_6) + \mathrm{Rec}(0, \Delta_2, \Delta_3, \Delta_4, 0, 0)$$

$$= g + \Delta$$

So  is limited to algebraic manipulation!

Linear Secret Sharing

Need: A way to store data on $\Sigma(G)$ so that algebraic manipulation can be detected.



- Privacy of SS $\Rightarrow$ Privacy of $\Sigma(G)$.
- Linearity of SS $\Rightarrow$ Algebraic Manipulation.

Algebraic Manipulation Detection (AMD) Codes

- An AMD Code consists of
  – A probabilistic encoding function E: $\boldsymbol{S} \to \boldsymbol{G}$
  – A decoding function D: $\boldsymbol{G} \to \boldsymbol{S} \cup \{\perp\}$
- For any s, $\boldsymbol{D}(E(\boldsymbol{s})) = \boldsymbol{s}$
- For any $\boldsymbol{s} \in \boldsymbol{S}$, $\Delta \in \boldsymbol{G}$

$$\Pr[\boldsymbol{D}(E(\boldsymbol{s}) + \Delta) \notin \{\boldsymbol{s}, \perp\}] \leq \epsilon$$

- Robust Secret Sharing: Share $\boldsymbol{E}(\boldsymbol{s})$.

Robust Linear Secret Sharing

Recall:

$$g' = \mathrm{Rec}(S_1, S'_2, S'_3, S'_4, S_5, S_6) = g + \Delta$$

$$= E(s) + \Delta$$

$$\Downarrow$$

$$D(g') = D(E(s) + \Delta) \in \{s, \perp\}$$

Construction of AMD Code

$$\boldsymbol{E}(\boldsymbol{s}) = (\boldsymbol{s},\ \boldsymbol{k},\ \boldsymbol{k}^{d+2} + \sum_{i=0}^{d} \boldsymbol{s}_i \boldsymbol{k}^i)$$

where $\boldsymbol{k}$ is random.

Parameters and Optimality

- To get robustness security $\epsilon = 2^{-\kappa}$, encoding of s adds overhead $2\kappa + O(\log(|s|))$ bits.
- Almost matches lower bound of 2κ bits.
- Previous constructions of Robust SS implicitly defined AMD codes with overhead linear in $|s|$.
  [CPS02, OK96, PSV99]
- To share a 1MB message with robustness $\epsilon = 2^{-128}$
  - Previous construction had an overhead of 2 MB.
  - We get an overhead of less than 300 bits.

Application #2

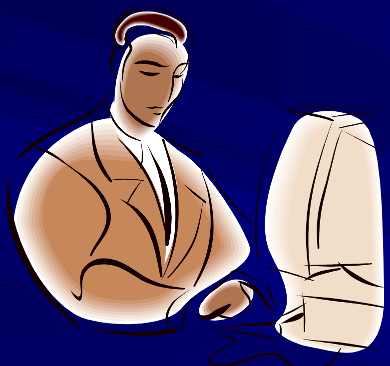
Fuzzy Extractors

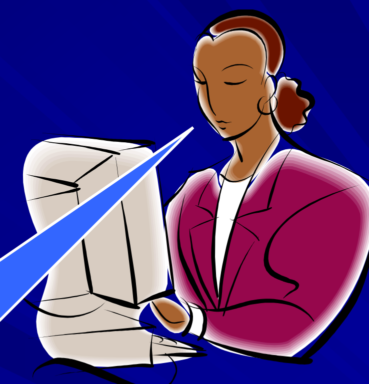
<u>**Robust**</u> Fuzzy Extractors

Secret w:
"secret_Password"

$w \approx w'$

Secret w':
"Secret~password"





$w \rightarrow$ Gen $\rightarrow P$, R

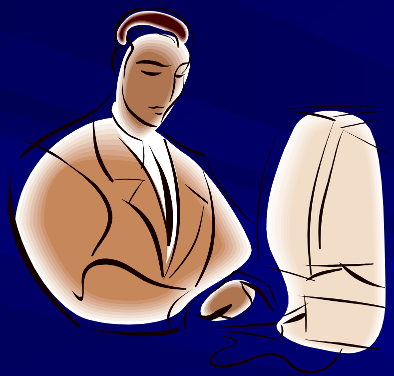
P^*, $w' \rightarrow$ Rep $\rightarrow R^*$

<u>Robust</u> fuzzy extractor:
I will detect $P^* \neq P$.

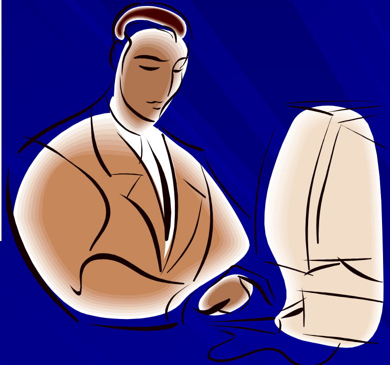
P.

Robust Fuzzy Extractors

Secret w



Secret w'



(Bob in the future)

$w \rightarrow$ Gen $\rightarrow P, R$

$P^*, w' \rightarrow$ Rep $\rightarrow R/\perp$

Does not allow interaction!

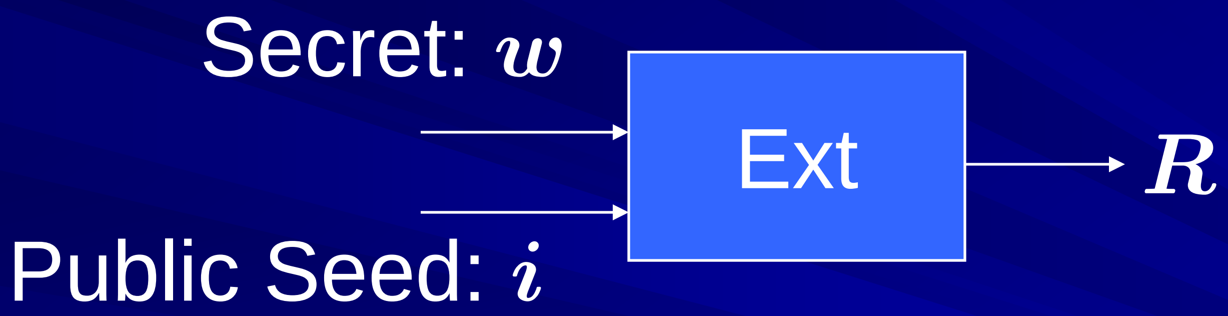
The Price of Robustness

- Non-robust fuzzy extractors with “good” parameters were constructed for several natural metrics. [DORS04]
- Until now, to get robustness, you had to choose:
  - Interaction + computational assumptions + CRS [BDKOS05]
  - Random Oracle model [BDKOS05]
  - Entropy rate of w more than ½ + extract short keys [DKRS06]
- Would like to get a non-interactive protocol that works for all entropy rates and does not require random oracles.

I.T. robustness requires that the entropy rate of w is more than ½, even in the non-fuzzy case w=w’.
  - The price of robustness, w.o. RO/CRS/assumptions, is **HIGH**! [DS02]

This talk: Robustness is essentially **FREE** in the CRS model!

Randomness Extractors



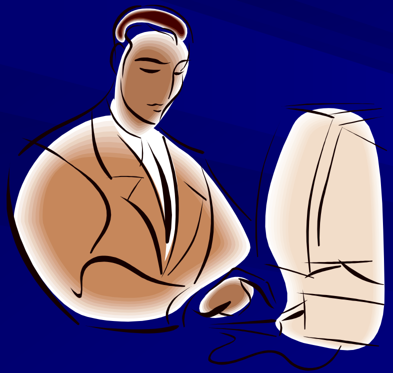
Can extract almost all entropy in w.
The extracted string is random, even given
The public seed

$$(i, R) \approx (i, U)$$

Non-fuzzy Key Exchange

Not robust!

Choose i



i

R=Ext(w,i)

R=Ext(w,i)

Non-fuzzy Key Exchange

CRS: Extractor seed i

Robust!

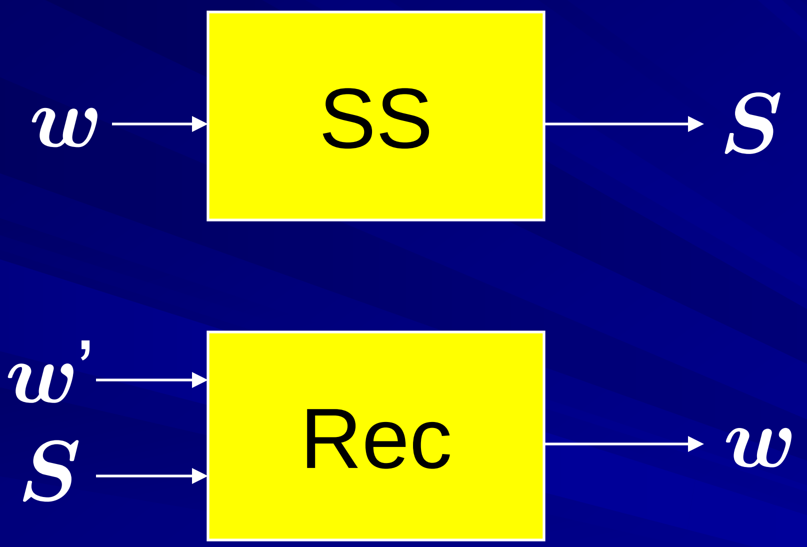
R=Ext(w,i)

R=Ext(w,i)

Trivial! No communication necessary!

But does not generalize to fuzzy case…

Correcting Errors using a Secure Sketch



SS(w) is very short and does not leak out much info about w.

Fuzzy Extractor

w

Cannot put in CRS since S is user-specific.

w'

$P = (i, S)$

Gen() : Rep(w', P) :

$w \rightarrow$, $i \rightarrow$ **Ext**

$w' \rightarrow$ **Rec** $\rightarrow w$

Let's try to only put i in the CRS and "authenticate" S.

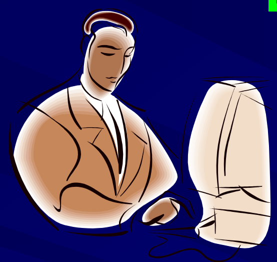
$w \rightarrow$ **SS** $\rightarrow S$

$w \rightarrow$, $i \rightarrow$ **Ext** $\rightarrow R$

Robust Fuzzy Extractor?

CRS: Extractor seed i

w



w'

$P = (S, \sigma)$ →

Gen():

Rep(w', P):

w → Ext → $R,$ R k
i →

w → SS → S

k → MAC → σ
S →

Use k as a key to a MAC to "authenticate" S

w' → Rec → w
S →

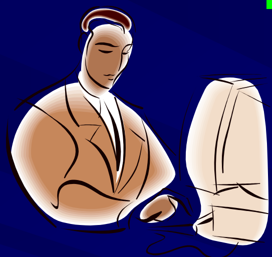
w → Ext → R R k
i →

k → Ver → "yes", "no"
S →
σ →

Robust Fuzzy Extractor?

CRS: Extractor seed i

w



w'

$P^* = (S^*, \sigma^*)$ $\longrightarrow$

Gen():

$w, i \rightarrow$ Ext $\rightarrow R, k$

$w \rightarrow$ SS $\rightarrow S$

$k, S \rightarrow$ MAC $\rightarrow \sigma$

Rep(w',P):

$w', S^* \rightarrow$ Rec $\rightarrow w$

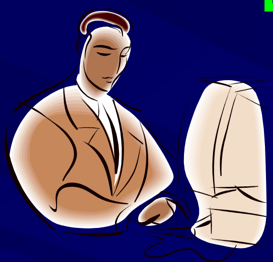
$w, i \rightarrow$ Ext $\rightarrow R, k$

$k, S^*, \sigma^* \rightarrow$ Ver $\rightarrow$ "yes", "no"

Robust Fuzzy Extractor?

$\boxed{\text{CRS: Extractor seed } i}$

w



w'

$P^* = (S^*, \sigma^*)$ $\longrightarrow$

Gen():

$w, i \rightarrow$ Ext $\rightarrow R, k$

$w \rightarrow$ SS $\rightarrow S$

$k, S \rightarrow$ MAC $\rightarrow \sigma$

Rep(w',P):

$w', S^* \rightarrow$ Rec $\rightarrow \boxed{w^*}$

$\boxed{w^*}, i \rightarrow$ Ext $\rightarrow R, k$

$k, S^*, \sigma^* \rightarrow$ Ver $\rightarrow$ "yes", "no"

Might not be secure!
But let’s see - how insecure is it?
Assume that the secure sketch
and extractor are linear...

Robust Fuzzy Extractor?

CRS: Extractor seed i

w

w'

$$\xrightarrow{P^* = (S^*, \sigma^*)}$$

Gen():

$$w, i \longrightarrow \boxed{\text{Ext}} \longrightarrow R, k$$

$$w \longrightarrow \boxed{\text{SS}} \longrightarrow S$$

$$k, S \longrightarrow \boxed{\text{MAC}} \longrightarrow \sigma$$

Rep(w', P):

$$w', \boxed{S+\Delta_S} \longrightarrow \boxed{\text{Rec}} \longrightarrow w^*$$

$$w^*, i \longrightarrow \boxed{\text{Ext}} \longrightarrow R, k^*$$

$$k^*, S^*, \sigma^* \longrightarrow \boxed{\text{Ver}} \longrightarrow \text{“yes”, ”no”}$$

Robust Fuzzy Extractor?

CRS: Extractor seed i

w

w'

$P^* = (S^*, \sigma^*)$

Gen():

$w \rightarrow$ Ext $\rightarrow R, k$
$i \rightarrow$ Ext

$w \rightarrow$ SS $\rightarrow S$

$k \rightarrow$ MAC $\rightarrow \sigma$
$S \rightarrow$ MAC

Rep(w',P):

$w' \rightarrow$ Rec $\rightarrow \boxed{w+\Delta_w}$
$S+\Delta_S \rightarrow$ Rec

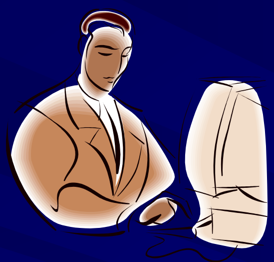
$\boxed{w+\Delta_w} \rightarrow$ Ext $\rightarrow R, k^*$
$i \rightarrow$ Ext

$k^* \rightarrow$ Ver $\rightarrow$ “yes”, ”no”
$S^* \rightarrow$ Ver
$\sigma^* \rightarrow$ Ver

Robust Fuzzy Extractor?

CRS: Extractor seed i

w



$P^* = (S^*, \sigma^*)$

w'

Gen():

Rep(w', P):

$w \rightarrow$ Ext $\rightarrow R, k$
$i \rightarrow$ Ext

$w \rightarrow$ SS $\rightarrow S$

$k, S \rightarrow$ MAC $\rightarrow \sigma$

$w' \rightarrow$ Rec $\rightarrow w+\Delta_w$
$S+\Delta_S \rightarrow$ Rec

$w+\Delta_w \rightarrow$ Ext $\rightarrow R$, $\boxed{k+\Delta_k}$
$i \rightarrow$ Ext

$\boxed{k+\Delta_k}$, S^*, $\sigma^* \rightarrow$ Ver $\rightarrow$ "yes", "no"

Robust Fuzzy Extractor?

- Can think of MAC key k as stored on a device $\Sigma(G)$.
- Can't encode k using an AMD code.
- Need a new MAC primitive.

MAC with Key Manipulation Security (KMS-MAC)

- A (one-time) MAC that is secure even if the key used for verification is stored on $\Sigma(G)$.

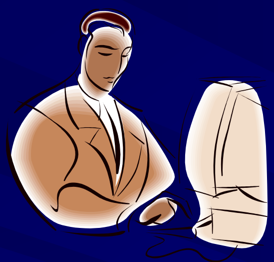
- Given σ = MAC$_k(s)$ can't come up with Δ and σ' = MAC$_{k+\Delta}(s')$.

- Systematic AMD code $\Rightarrow$ KMS-MAC:
  - $E(s) = (s, k, h(s,k))$
  - MAC $_{(k_1,k_2)}(s) = h(s,k_1)+k_2$

Use a KMS-MAC!

CRS: Extractor seed i

w



w'

$P^* = (S^*, \sigma^*)$

Gen():

$w, i \to$ Ext $\to R$

$w \to$ SS $\to S$

$k, S \to$ MAC $\to \sigma$

Rep(w', P):

$w', S^* \to$ Rec $\to w^*$

$w^*, i \to$ Ext $\to R$

k

$\Sigma(G)$

$k, S^*, \sigma^* \to$ Ver $\to$ “yes”, ”no”

Parameters

- Because our KMS-MAC has short keys, we loose very little randomness to achieve robustness!
- In the CRS model, robustness comes essentially for FREE.
  - At least for “linear” fuzzy extractors

Review

Devices $\Sigma(G)$ appear naturally in crypto applications.

- Linear Secret Sharing.
- Fuzzy Extractors in CRS model.

Use AMD codes or KMS-MACs to get robustness.

THANK YOU!

Questions?