On Extractors, Error-Correction and Hiding All Partial Information

Yevgeniy Dodis

New York University

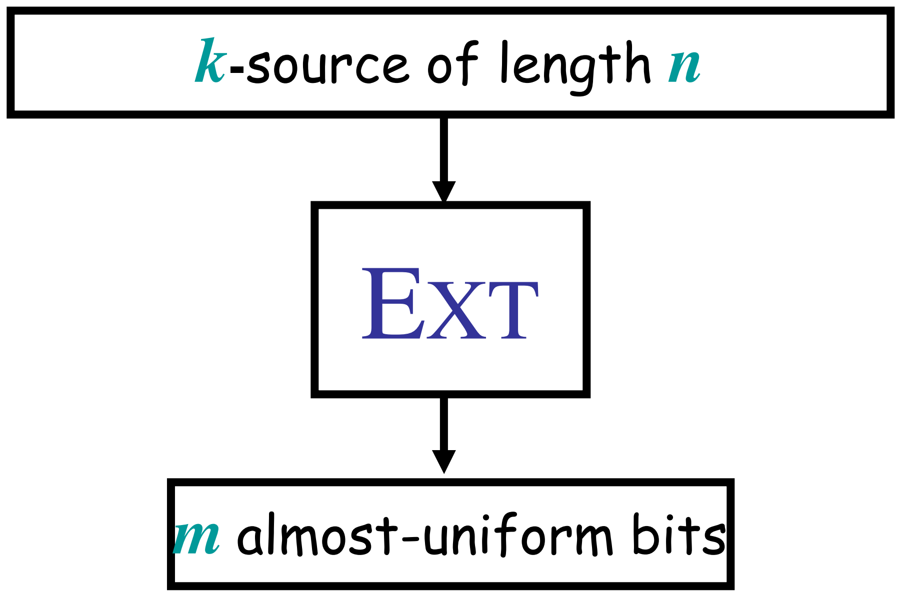
Based on several joint works with the following co-authors:
Xavier Boyen, Jonathan Katz, Rafail Ostrovsky, Leonid Reyzin and Adam Smith

Imperfect Random Sources

- Randomness is crucial in many areas
  - Especially cryptography (i.e., secret keys)
- Usually, assume a source of truly random bits
- However, often deal with imperfect randomness
  - Physical sources
  - Biometric data
  - Partial knowledge about secrets
- Necessary assumption: must have (min-)entropy
  - (Min-entropy) *k*-source: $\Pr[X=x] \le 2^{-k}$, for all x
- Can we extract (nearly) perfect randomness from such realistic, imperfect sources?

Extractors: 1st attempt

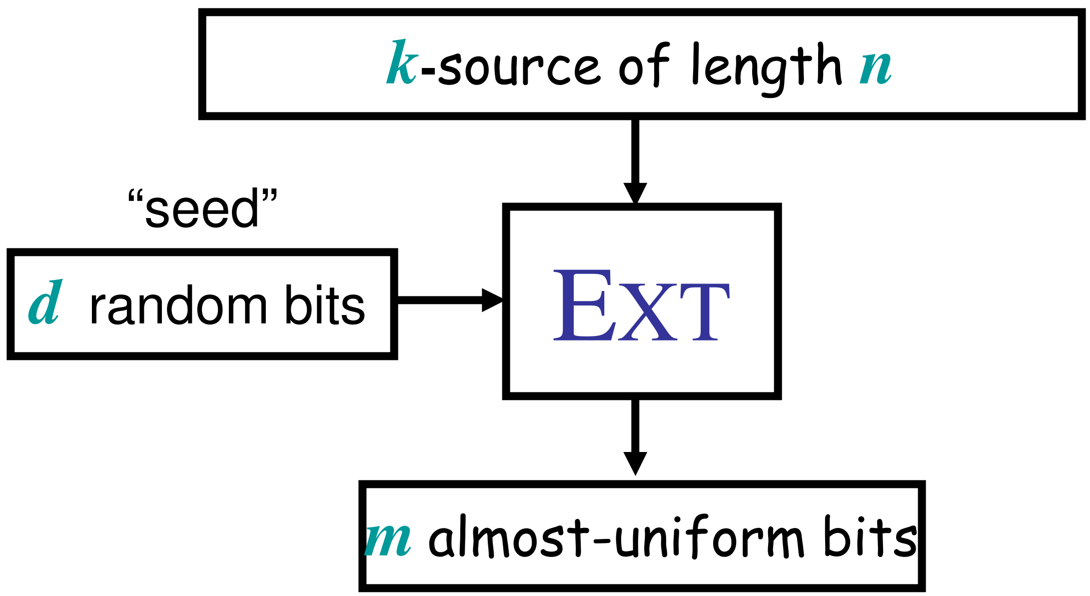
- A function $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}^m$ such that $\forall$ k-source X, $\mathrm{Ext}(X)$ is "close" to uniform.



- Impossible! $\exists$ set of 2^{n-1} inputs x on which first bit of $\mathrm{Ext}(x)$ is constant $\Rightarrow$ "flat" $(n\text{-}1)$-source X, bad for Ext.

Modern Extractors [NZ]

- Def: (k,ε)-extractor is $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ s.t. $\forall$ k-source X, $\mathrm{Ext}(X,U_d)$ is ε-close to U_m.



- Key point: **seed can be *much* shorter than output.**
- Goals: minimize seed length, maximize output length.

Strong Extractors

- Output looks random even after seeing the seed
  - Very handy in some applications !
  - Ex: only "remember" biometric secret X, publish seed I is and use $\mathrm{Ext}(X, I)$ as the "effective" secret key.
- Def: Ext is a (k,ε) strong extractor if

  $\mathrm{Ext}'(x;i) = i \circ \mathrm{Ext}(x, i)$ is a (k,ε)-extractor

- Optimal: $d \approx \log(n\text{-}k) + \log(1/\varepsilon)$, $m \approx k - 2\log(1/\varepsilon)$
  - In many crypto applications, OK to have $d = O(n)$

Leftover Hash Lemma

- Universal Hash Family $\{ h_i : \{0,1\}^n \rightarrow \{0,1\}^m \}$:

$$\forall x \neq y, \ \ \Pr_I(h_I(x) = h_I(y)) = 2^{-m}$$

- Leftover Hash Lemma [HILL]: universal hash functions $\{h\}$ yield strong extractors: $(I, h_I(X)) \approx_\varepsilon (I, U_m)$
  - optimal output length: $m = k - 2\log(1/\varepsilon)$
  - seed length: $d = O(n)$
- Ex: $\mathrm{Ext}(x;a)$ = first m bits of $a \cdot x$ in $GF(2^n)$
- Many generalizations known (stay tuned !)

Aren't We Cheating?

- Need truly random seed to extract randomness??
  - Remember, extract much more than invest !
  - In some applications have "local randomness"
  - Sometimes go over all seeds for derandomization
- Indeed, many applications !
  - Derandomization [Sip,GZ,MV,STV,NZ,INW,RR,GW,….]
  - Distributed and Network Algorithms [WZ,Zuc,RZ,Ind]
  - Hardness of Approximation [Zuc,Uma,MU]
  - Data Structures [Ta]
  - Pseudorandom number generation [BH]
  - Cryptography !
    [CDHKS,DSS,KZ,GRS,MW,Lu,Vad,Din,DS1,DS2,DRS,BDKOS…]

When to Use Extractors?

- The obvious usage is for extracting good randomness (key derivation)
- Less known: for arguing privacy !

1. Output of extractor hides the actual distribution on X
2. [DS1]: in fact, it "hides every deterministic function of X" !

- Some applications need both usages !

Entropic Security [CMR,RW]

- A map S() is called (k,ε)-entropically secure if $\forall$ k-source X, $\forall$ predictors, $\exists$ simulator, $\forall$ functions f, seeing S(X) "does not help":

$$\Pr\left[S(X) \rightarrow \boxed{\text{Predictor}} \rightarrow f(X)\right] \leq \Pr\left[\boxed{\text{Simulator}} \rightarrow f(X)\right] + \varepsilon$$

- Also say S() hides all functions of X
- Notice, S() must be probabilistic (f = S)
- S() must also be one-way (f = identity)
- Identical to semantic security [GM], but for high-entropy distributions

Comparing to Shannon

- Shannon Security: $S(X)$ is independent of X
  - Very strong, hides all "a-posteriori" functions
  - As such, $S(X)$ can't be "useful" for anything
- E-security "only" hides "a-priori" functions
  - Can leak "useless" info while still being "useful"
- Equivalent without min-entropy constraint
- Warning: E-security does not compose well
  - Like most i.t. notions, can only be used once (e.g., $S(X;r_1)$, $S(X;r_2)$ might potentially leak X)

High-Entropy Indistinguishability

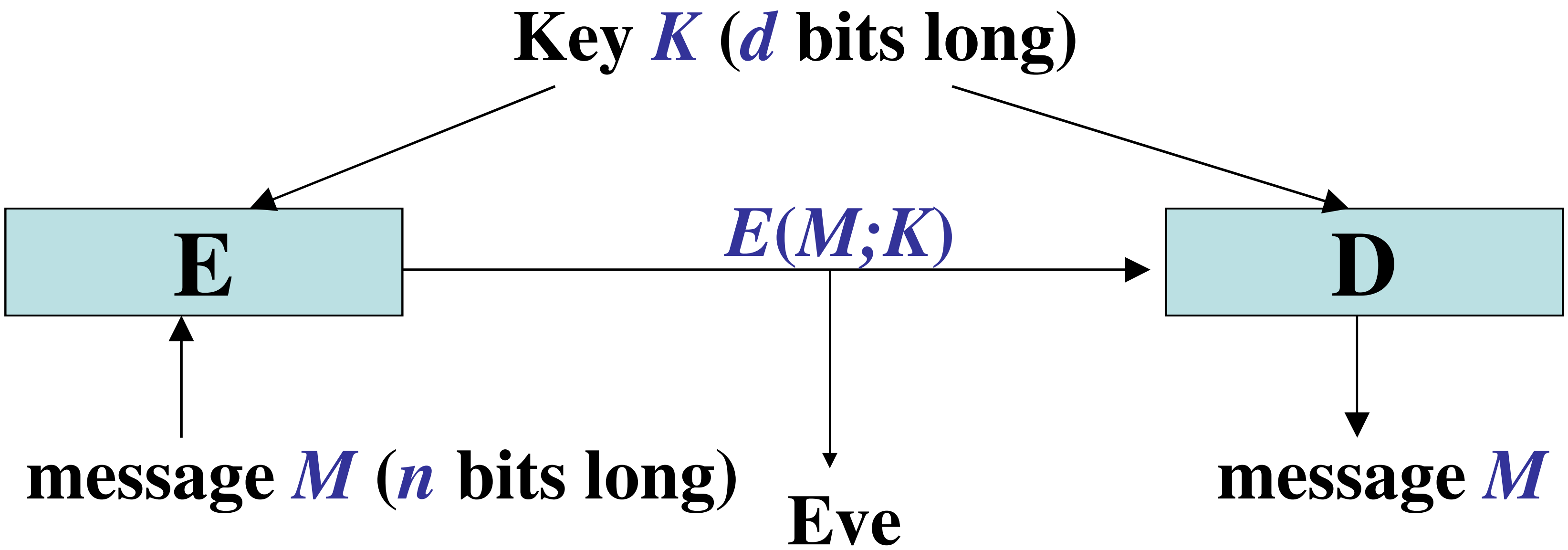
- A map S() is called (k,ε)-indistinguishable if $\forall$ k-sources X, Y, S(X) is ε-close to S(Y)
  - In particular, all of them are ε-close to S(U)
  - (k,ε)-extractors are also $(k,2\varepsilon)$-indistinguishable
- Thm [DS1]: If S() is (k,ε)-indistinguishable then it is $(k+2,8\varepsilon)$-entropically secure
- Corollary: extractors for min-entropy k hide all functions for sources of min-entropy $k+2$
- Punchline: to argue entropic security, enough to construct a "special-purpose" extractor

"Special-Purpose" Extractors

- Sometimes, plain extractors are not enough!
  - Need extractors with "extra properties"
- Scenario 1: more robust key derivation
  - Local computability (bounded storage model)
  - Noise-tolerance (biometrics)
- Scenario 2: when extraction is merely a convenient tool for arguing entropic security
  - Invertibility (for encryption)
  - Collision-resistance (for hash functions)
  - Error-correction (for information-reconciliation)
  - Unforgeability (for message authentication)
- Scenario 3: combination of scenarios 1 & 2

Adding Invertibility: Entropically-Secure Encryption

Symmetric Encryption



- Shannon: Symmetric Encryption without computational assumptions requires $d \geq n$ (achieved by one-time pad)
- Russell and Wang [RW]: What can be said when the message is guaranteed to have high entropy?

Entropically-Secure Encryption

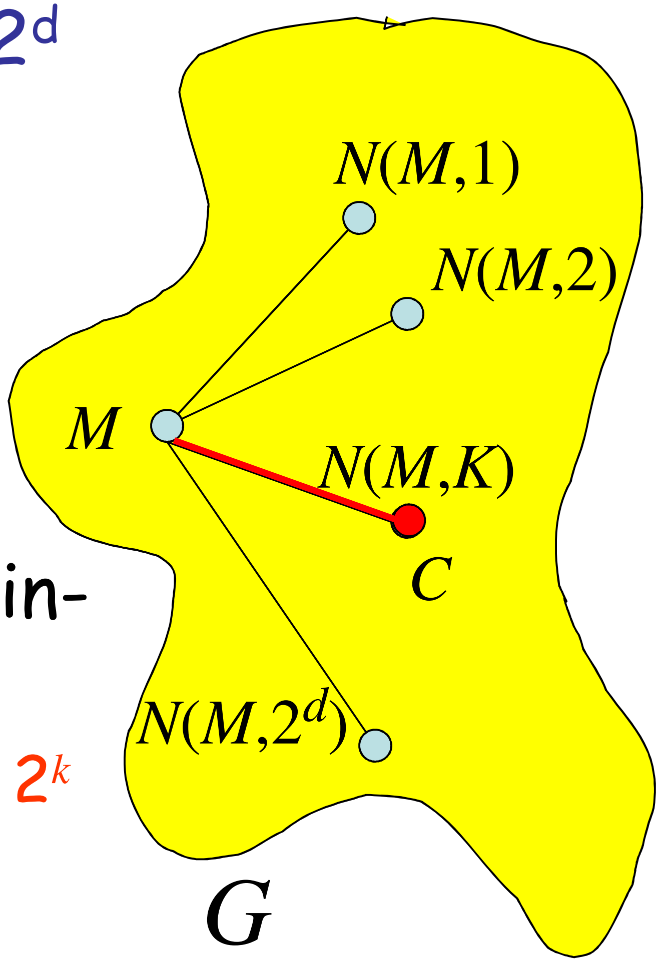
- Require E to be (k,ε)-entropically secure
  - Ciphertext hides all functions of plaintext
  - Note: Shannon security corresponds to $k = 1$
- [RW]: can beat Shannon's bound when $k > 1$
  - Pretty ad-hoc and complicated
- [DS1]: suffices to construct $E(M;K)$ which is an extractor for min-entropy $k-2$!
  - Leads to better (optimal !) constructions
  - Much simpler to understand/analyze than [RW]
- Thus, need (k,ε)-extractor whose source can be recovered from its output and its seed.

Invertible Extractors

- If $C = E(M; K)$, then we want
  1. $C \approx$ random, if K random and M has entropy k
  2. One can recover ("decrypt") M from C and K
  3. Goal: minimize $d = |K|$
- Note, $|C| \geq |M| = n$ (by invertibility)
- Also, C has $|C| \geq n$ bits of entropy (since it is random)
- Since M only has k bits of entropy, we must have key length $|K| \geq n - k$
- Can we achieve it???

Using Graphs for Encryption

- Graph on 2^n vertices of degree 2^d
- Consider $\mathbf{E}\ (M,K) = N\ (M,K)$
  - Random step from M
  - Decryption assumes labeling is "invertible", which is easy to get (Cayley graphs)
- <u>Goal</u>: get to uniform from any min-entropy $\geq k$ distribution on M
  - Expansion ! Want any set of size $\geq 2^k$ to expand to all vertices in 1 step!
- Can achieve $d = n - k + 2\log(1/\varepsilon)$ (using the Ramanujan expanders)



Sparse One-Time Pad

- For r.v. X over $\{0,1\}^n$ and $\alpha \in \{0,1\}^n$, let
  $\text{bias}_\alpha(X) = 2(\Pr[\alpha \odot X = 0] - \frac{1}{2}) = \mathbb{E}[(-1)^{\alpha \odot X}]$
  - X is δ-biased if $|\text{bias}_\alpha(X)| \leq \delta$ for all $\alpha \neq 0$
  - Can sample δ-biased X with $2\log(n/\delta)$ bits
- Fact: If X is δ-biased, M is k-source then

  $M \oplus X \approx_\varepsilon$ **uniform**, where $\varepsilon = \delta \cdot 2^{(n-k)/2}$
- Use optimal δ-biased sets and get "sparse one-time pad" with $d = n - k + 2\log(n/\varepsilon)$

Probabilistic One-Time Pad

- Modified LHL:
  - $E(M; K) = (\, I,\ M \oplus h_I(K)\,)$
  - probabilistic encryption (I is not part of K)
  - Here $\{h_i : \{0,1\}^d \to \{0,1\}^n\}$ is "XOR-universal":

$$\forall a \in \{0,1\}^n, x \neq y,\ \Pr_I(h_I(x) \oplus h_I(y) = a) \ = \ 2^{-n}$$

- **LHL' [new]:** If $\{h_i\}$ is XOR-universal and $k \ \geq n - d + 2\log(1/\varepsilon)$ then

$$(\, I\,, M \oplus h_I(K)\,) \approx_{\varepsilon} (\, I\,, U_n)$$

Probabilistic one-time pad: $d = n - k + 2\log(1/\varepsilon)$

Invertible Extractors

- Theorem [DS1]: three constructions
  - From expander graphs, achieve optimal $d = n - k + 2\log(1/\varepsilon)$, where ε is the "error"
  - "Sparse One-time Pad: $E(M;\ K) = M \oplus S(K)$, where $d = n - k + 2\log(n/\varepsilon)$
    - $S(K)$ is a point sampled from $(\varepsilon \cdot 2^{(k-n)/2})$-biased set
  - "Probabilistic OTP": get $d = n - k + 2\log(1/\varepsilon)$
    - $E(M;\ K) = (\ I,\ M \oplus h_I(K)\ )$
    - probabilistic encryption (I is not part of K)
    - Here $\{h_i : \{0,1\}^d \to \{0,1\}^n\}$ is "XOR-universal"

Adding Collision-Resistance: Perfectly One-Way Hash Functions

Collision-Resistant Extractors

- Collision: $(w,i) \neq (w',i')$ s.t. $\mathrm{Ext}(w;i) = \mathrm{Ext}(w';i')$
  - Strong extractors: i, $w \neq w'$ s.t. $\mathrm{Ext}(w;i)=\mathrm{Ext}(w';i)$
- "Commit" to w by publishing $(i, \mathrm{Ext}(w;i))$
  - Great decommitment: simply present w !
- <u>Entropic Security</u>: if entropy of W is at least k, then $(I, \mathrm{Ext}(W;I))$ hides all functions of W (weaker than usual hiding)
- Note: don't need full power of extractors, suffices to have (k,ε)-indistinguishability

Construction

- Yet another variant of LHL:
  - $\mathrm{Ext}(W ; I) = f(h_I(W))$
  - $f: \{0,1\}^N \rightarrow \{0,1\}^m$ is arbitrary function
  - $\{h_i : \{0,1\}^n \rightarrow \{0,1\}^N\}$ are pairwise independent:

$$\forall x \neq y,\ (h_I(x), h_I(y)) \equiv (U_N, U_N)$$

- **LHL" [DS2]:** If $\{h_i\}$ is pairwise independent and $k \geq m + 2\log(1/\varepsilon)$ then

$$(I, f(h_I(W))) \approx_\varepsilon (I, f(U_N))$$

(gives an extractor if $f(U_N)$ is uniform)

Construction

- **LHL'':** If $\{h_i\}$ is pairwise independent and $k \geq m + 2\log(1/\varepsilon)$ then

$$(I, f(h_I(W))) \approx_\varepsilon (I, f(U_N))$$

- Apply with f = CRHF and family of pairwise independent permutations (e.g., $\{ax+b | a \neq 0\}$)
  - Permutations ensure collision-resistance
- Gives Perfectly One-Way Hash Functions and Obfuscators for Equality for inputs with entropy > output of CRHF + 2log(1/ε)

Adding Locally Computable Aspect: Key Derivation in Bounded Storage Model

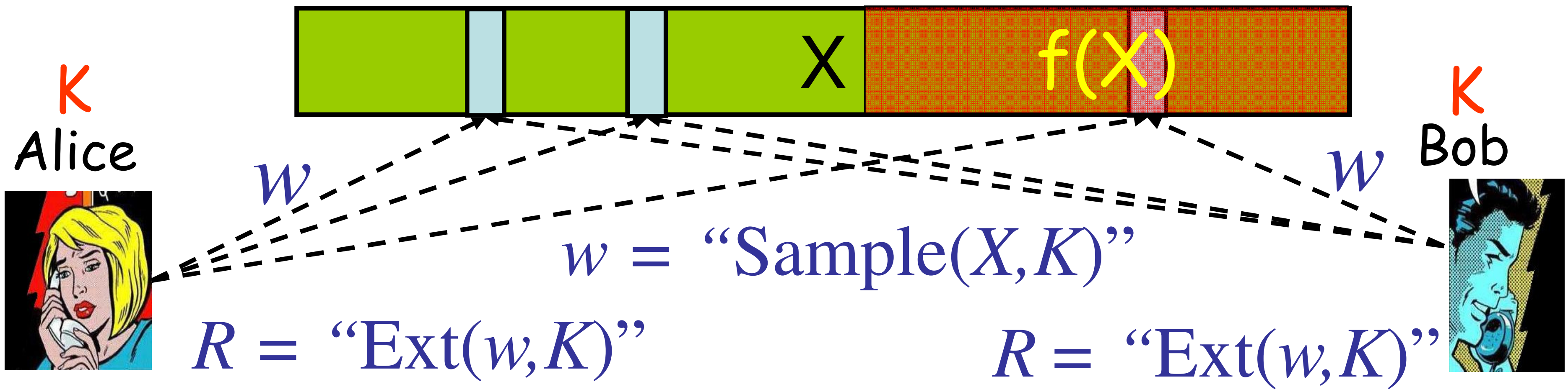
Bounded Storage Model [Mau]

- <u>Setting</u>:
  - Alice and Bob share a short random key K (have local randomness, although not needed)
  - A huge random (high entropy enough) string X of length N is broadcast to them
  - Eve is allowed to store any function $Z = f(X)$ of length γN, for some $\gamma < 1$
  - Thus, from Eve's perspective, X is imperfect, although still has high entropy

Bounded Storage Model

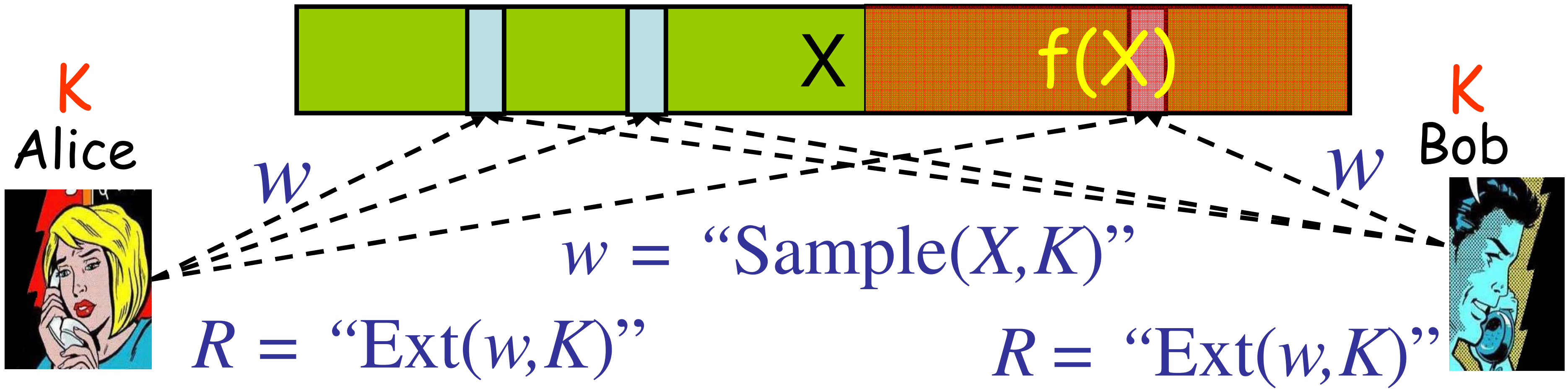
- Goal 1: Key Agreement
  - extract a much longer random key R from X using K
  - R is secret from Eve, for any storage function f
- Goal 2: Key Reuse
  - keep using the same K with subsequent (new) X's
- Goal 3: Everlasting security
  - R should be secure even if K is leaked later
- Simple solution: apply a strong extractor to X with seed K
- Satisfies goals 1-3, but requires Alice and Bob to read the entire X, which even Eve cannot do ☹ !

Locally Computable Extractors



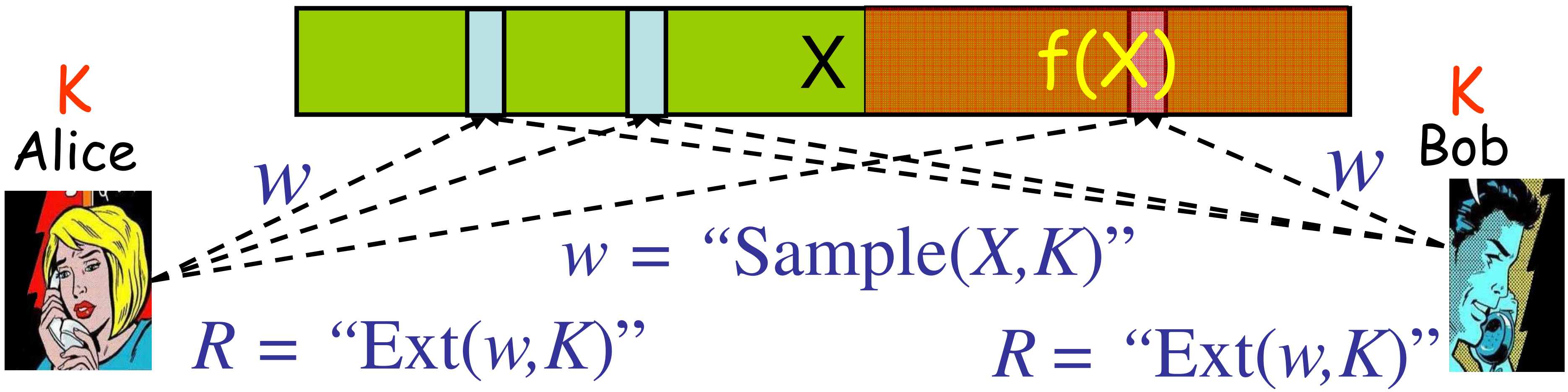
- Example [AR]:
  - K consists of t random indices $i_1,\ldots,i_t \in \{1..N\}$
  - w = $X[i_1] \ldots X[i_t]$, extract bit $R = w_1 \oplus \ldots \oplus w_t$
  - Can argue secure if $\gamma < 1/5$ and t "large enough"
  - Rate inefficient, but illustrates the point (indeed, improved by [DM, Lu, Vad])

Locally Computable Extractors



- "Sample-then-Extract" [Lu,Vad]
  - $K = (K_s, K_e)$, K_s & K_e – sampling & extraction keys
  - Use K_s to sample small subset of bits w from X
  - If "good" K_s is used, w still has high min-entropy from Eve's point of view
  - Use K_e as a seed to any good strong extractor

Locally Computable Extractors



- "Sample-then-Extract" [Lu,Vad]
  - $K = (K_s, K_e)$, K_s & K_e - sampling & extraction keys
- With optimal sampler and extractor:
  - can have key $|K| = O(\log N + \log 1/\varepsilon)$
  - extract m bits by reading $O(m)$ bits w from X

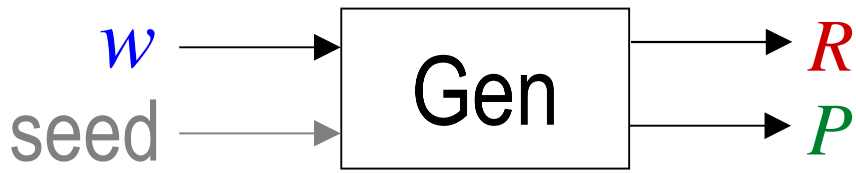
Adding Noise-Tolerance: Fuzzy Extractors and Secure Sketches

Biometrics

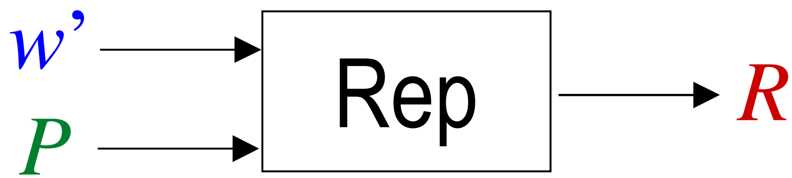
- Setting:
  - Want to use imperfect biometric data W as your secret key
  - Have local randomness, but can't "remember" it
- Simple Solution:
  - Apply strong randomness extractor
  - Store seed I for strong extractor in the public
  - Use Ext(W; I) as your "actual" secret key
- Problem: noisy nature of biometrics
  - Two different readings of W are likely to be different, although "close"

New Primitive: Fuzzy Extractor

- Reliably extract randomness out of w
- First time: generate random R from w (+ seed)

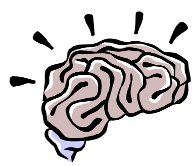
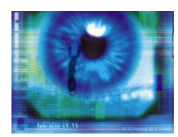


- Subsequently: reproduce R from P and any $w' \approx w$



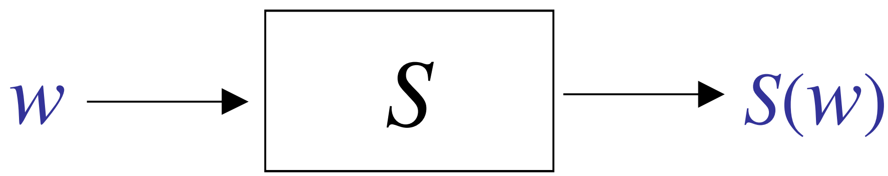
- R is nearly uniform given P if w has sufficient min-entropy (can put usual n, m, k, t, ε) distance
- Punchline: trade-off $|R| = m$ for error-tolerance (distance t) and non-uniformity (min-entropy k)

What does "Close" mean?

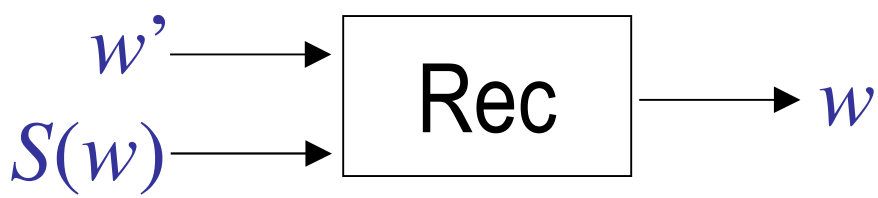
- Depends on the "natural" metric space for the underlying application!
  - Hamming Metric (feature-extraction systems)
  - Set Difference ("favorite" set in a large universe)
  - Edit Metric (handwriting / typing)
  - Permutation Metric (ranking-based preferences)
  - "Real" Metrics:     (complicated) 
- Different metrics require different techniques!
- [DORS]: General framework, specific algorithms

Building Block: Secure Sketch

- Add reliability by publicly storing sketch $S(w)$



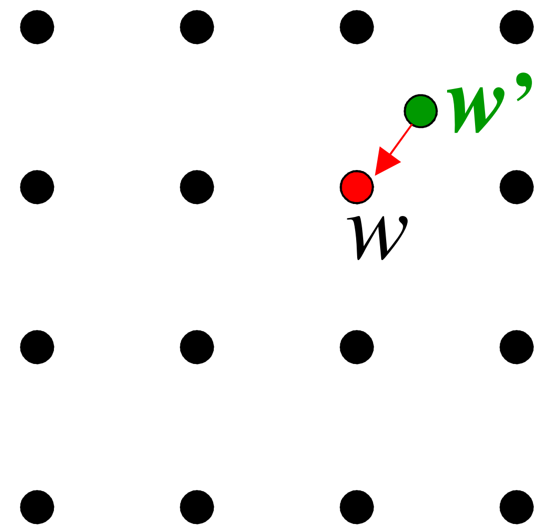
- Recover w from $S(w)$ and <u>any</u> $w' \approx w$ (w' close to w)



- w has "high" min-entropy even given $S(w)$
  - Entropy loss: how much entropy $S(w)$ revealed about w
  - Note, Entropy loss $\leq |S(w)|$ (good to have short sketch)
- <u>Punchline</u>: trade-off **entropy** for **error-tolerance**

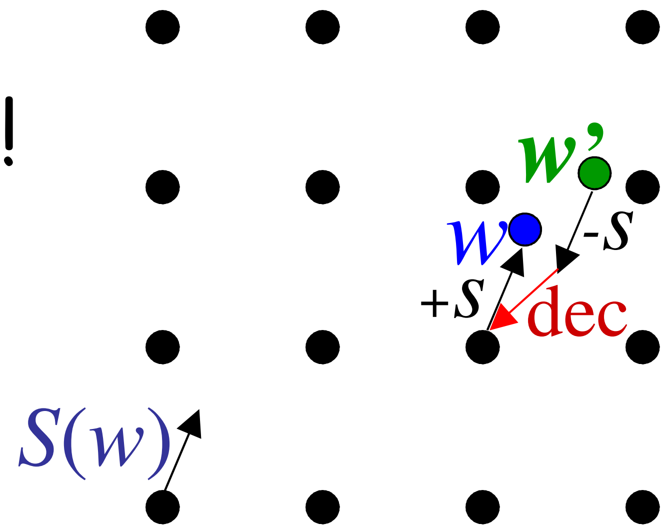
Secure Sketch in Hamming Space

- Idea: what if w is a codeword in an ECC?
- Decoding finds w from w'



Secure Sketch in Hamming Space

- Idea: what if w is a codeword in an ECC?
- Decoding finds w from w'
- If w not a codeword, simply shift ECC to contain w and just remember the shift !



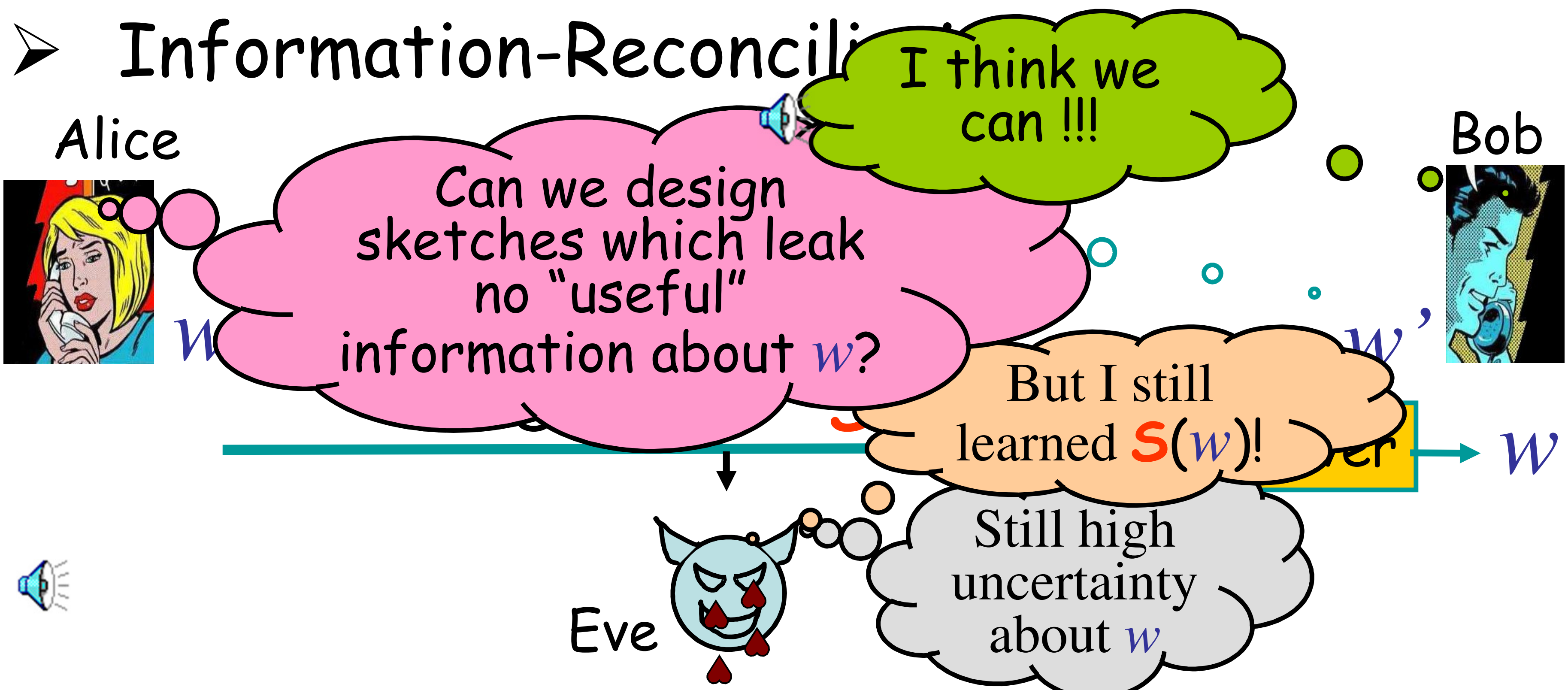
Code-Offset Construction

$S(w) = \text{syndrome}(w)$ OR $S(w;r) = w \oplus \text{ECC}(r)$

- If ECC expands a bits $\rightarrow$ n bits and has distance d:
  - Correct $t = d/2$ errors
  - $S(w)$ has $n - a$ bits $\Rightarrow$ entropy loss at most $n - a$
  - Optimal if code is optimal (sketch $\Rightarrow$ ECC)
  - Works for non-binary alphabets too (i.e., RS codes give optimal entropy loss = $2t \log q$)
- Appears in [BBR88, Cré97, JW02] under various guises
- [DORS]: also sketches for other metrics

Using Secure Sketches

- SS + strong extractor $\Rightarrow$ fuzzy extractor
  - Namely, set $P = (\, S(w), I \,)$, $R = \mathrm{Ext}(w\,;I\,)$
  - Extract $|R| \approx$ residual min-entropy – $2\log(1/\varepsilon)$
- Information-Reconcili

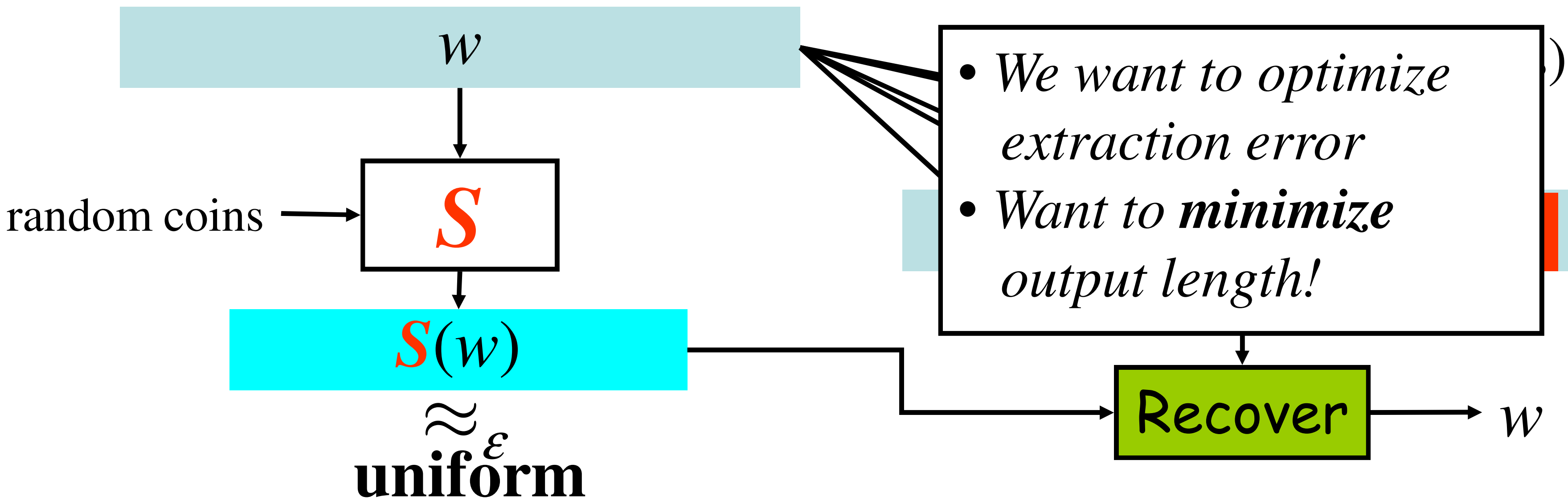


Correcting Errors Without Leaking Partial Information: Entropically-Secure Sketches

Entropically-Secure Sketches

- Design sketch $S(w)$ such that
  - Can recover w from $S(w)$ and any w' close to w
  - $S()$ is (k,ε)-entropically secure
- Notice, implies residual entropy $\geq \log(1/\varepsilon)$
- Converse false: code-offset leaked $\mathrm{syn}(w)$
- Suffices to construct (k,ε)-extractor which is also a sketch !
  - **Goal**: minimize number of "extracted" bits

Error-Correcting Extractors



Theorem [DS2]: If min-entropy $k = \Omega(n)$, then $\exists$ (strong) extractor $S(\cdot)$ (for Hamming errors) such that

- Can correct $t = \Omega(n)$ errors efficiently
- Error $\varepsilon = 2^{-\Omega(n)}$. In particular, $H_\infty(W \mid S(W)) = \Omega(n)$
- Output "only" $k\,(1-\Omega(1))$ bits

Compare with invertible extractors:

- not having $w' \approx w$ "forces" to extract $\geq n$ bits !

Error-Correcting Extractors

- Idea 1: Re... uniform,
  if X is ...
- Ide... ...ood
  sketch ... a good code

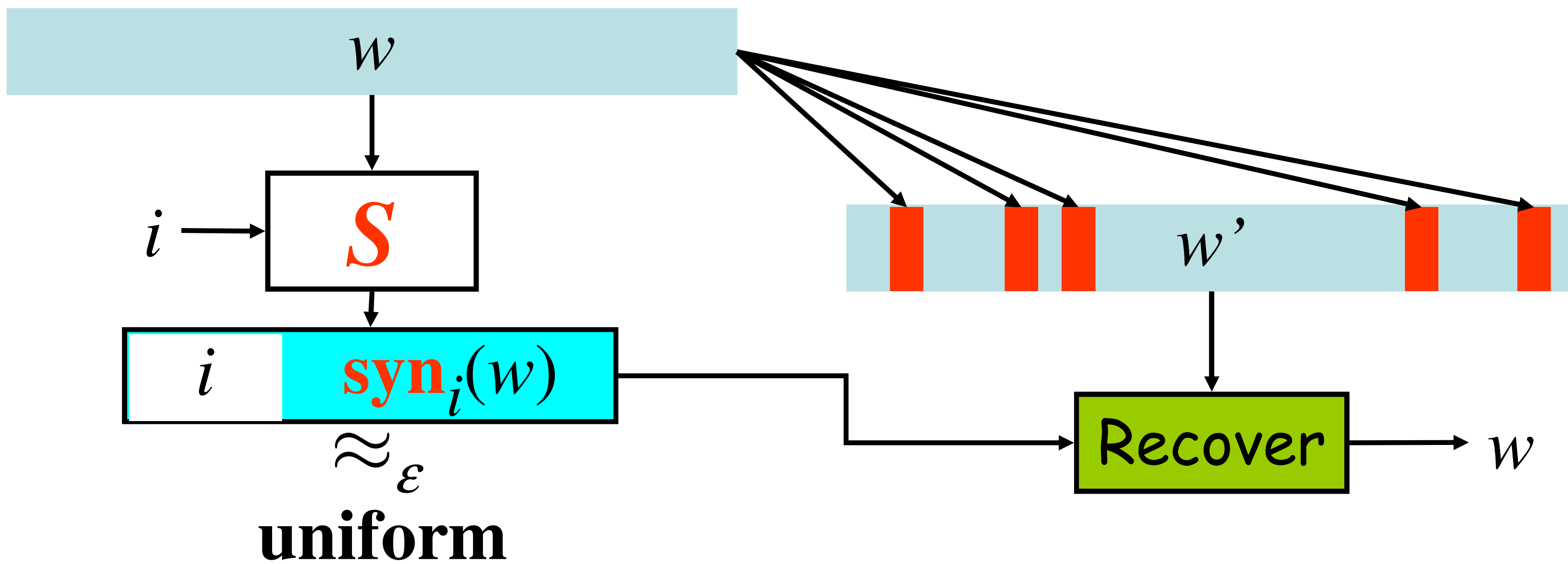
Recently constructed
by Shpilka'05
(bad params though)

- Can we achieve both simultaneously?
  - Yes for non-linear codes, but no explicit constructs ☹
  - No for linear codes (any α in the dual has $\alpha \odot X \equiv 0$) ☹
- Idea 3: use a family of (carefully chosen) linear codes to get the best of both worlds !

Construction

- Design family of codes $\{\mathrm{ECC}_i\}$ and set

$$\boxed{S(w;i) = (i, \mathrm{syn}_i(w)) \quad \text{OR} \quad S(w;i,r) = (i, w \oplus \mathrm{ECC}_i(r))}$$



Theorem [DS2]: There exist efficiently decodable codes with "needed parameters"

- for "large" alphabets get optimal parameters!

Construction

- Design family of codes $\{\mathrm{ECC}_i\}$ and set

$$\boxed{S(w;i) = (i, \mathrm{syn}_i(w)) \text{ OR } S(w;i,r) = (i, w \oplus \mathrm{ECC}_i(r))}$$

- <u>Theorem</u> [DS2]: If entropy $k = \Omega(n)$, there exists codes giving (strong) extractors s.t.
  - Can efficiently correct $t = \Omega(n)$ errors
  - Have (entropic) error $\varepsilon = 2^{-\Omega(n)}$
  - Output "only" $t\,(1-\Omega(1))$ bits
- Compare with invertible extractors:
  - not having $w' \approx w$ "forces" to extract $\geq n$ bits !

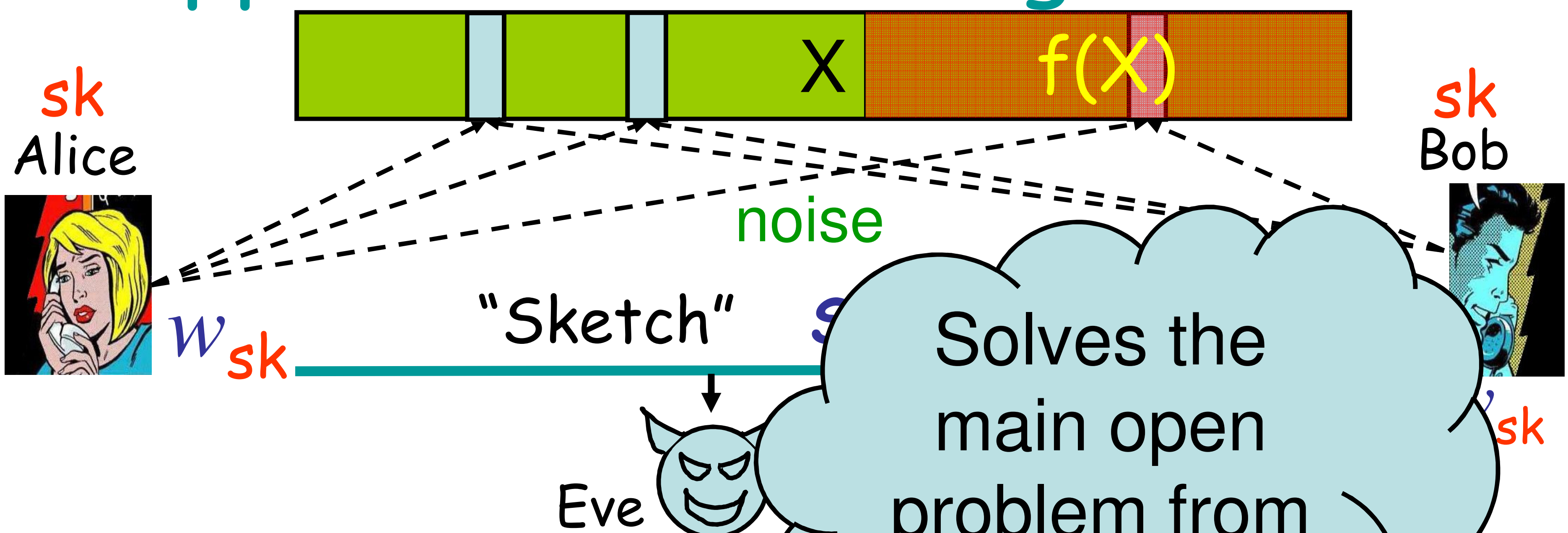
App: Private Fuzzy Extractors

- Recall, SS + strong extractor $\Rightarrow$ fuzzy extractor: set $P = (\ S(w), I\ )$, $R = \mathrm{Ext}(w\ ;I\ )$
  - Let's use "extractor-sketches" instead !
- Get FE where $(\boldsymbol{P}, \boldsymbol{R}) \approx_\varepsilon (\boldsymbol{U_1}, \boldsymbol{U_2})$
  - Even joint pair $(\boldsymbol{P}, \boldsymbol{R})$ hides all functions of $\boldsymbol{W}$!
- Called Private Fuzzy Extractors:
  - As opposed to usual fuzzy extractors, public data $\boldsymbol{P}$ does not reveal anything "useful" about the biometric $\boldsymbol{W}$, even if the key $\boldsymbol{R}$ is leaked !

App: Fuzzy POWHFs

- Recall, POWHFs allow to publish a value $\mathbf{Z}$ = "Commit(w)" s.t. given input w'
  - Verify($\mathbf{Z}$,w') accepts if and only if $w=w'$
  - Moreover, $\mathbf{Z}$ is (k,ε)-entropically secure
- What if want to test if distance(w,w') $< t$?
- Attempt: use secure sketch and publish ($\mathbf{Z}$, S(w))
  - Preserves collision-resistance ☺
  - Does not preserve entropic security ☹
- Solution: use entropically-secure sketch. Get
  - Fuzzy POWHFs
  - Equivalently, (weak) obfuscators for proximity queries

App: Bounded Storage Model



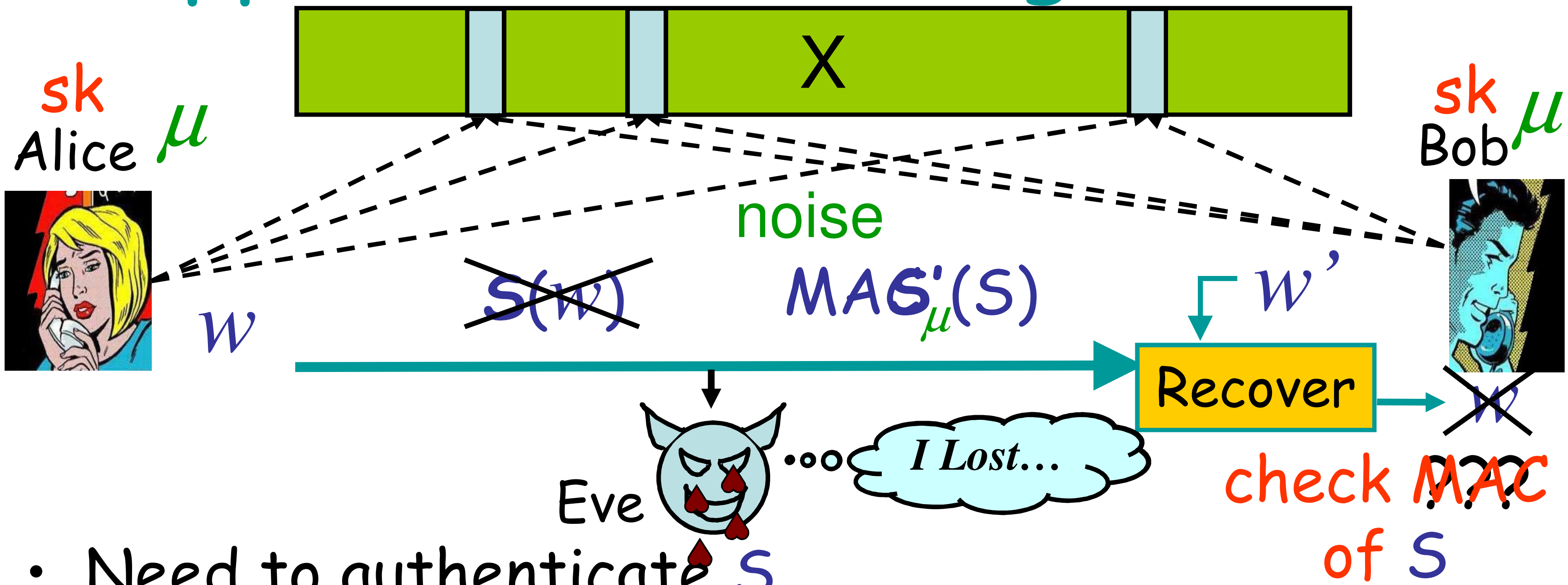
Solves the main open problem from [Ding'05]

- Shared secret samp…
- Goal: $H(W_{sk} \mid S(W_{sk}),$ s… …or Eve
- "Everlasting security": can we re-use sk?
- [Ding]: Not with usual sketches!
  - $S(W_{sk})$ leaks info on sk
- Extracting sketch: $\mathbf{S}(W_{sk1}) \approx \mathbf{S}(W_{sk2})$!

Adding Authentication:

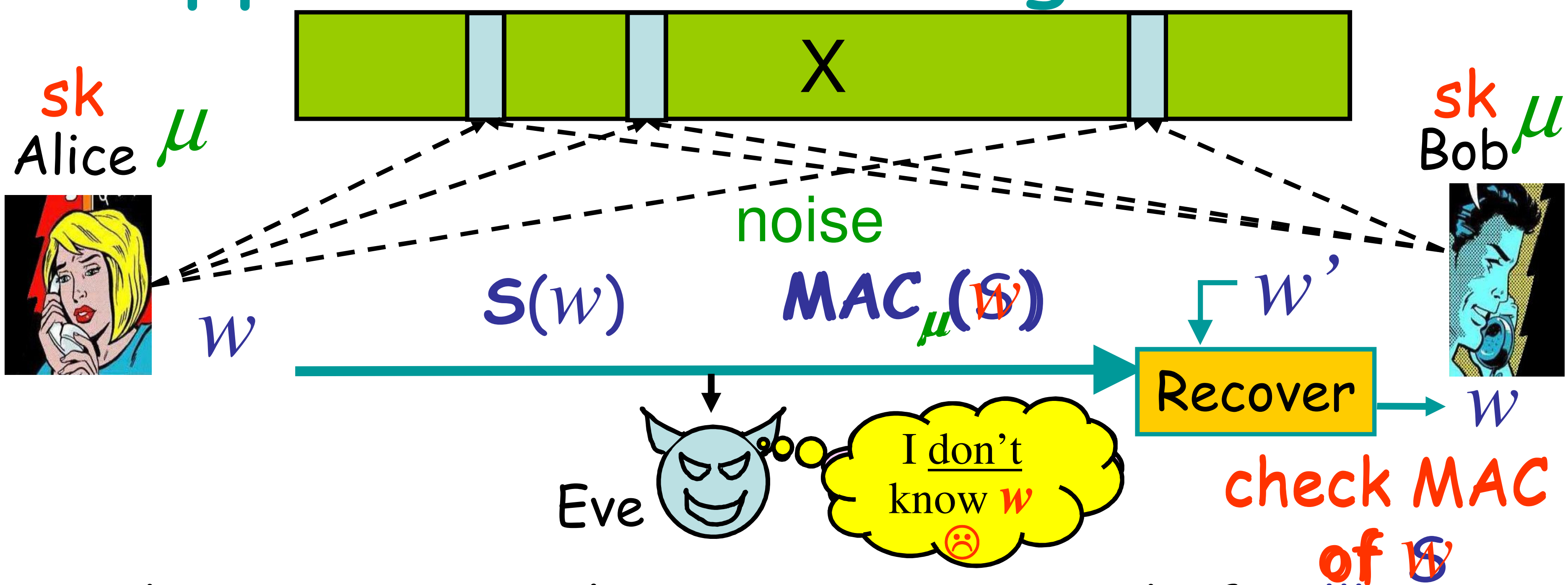
entropically-secure MACs, Robust FE/SS, ...

App: Bounded Storage Model



- Need to authenticate S
- No problem: add MAC key μ to sk
  - send MAC$_\mu$(S) together with S
- But which MAC???
  - Computational: lose information-theoretic security ☹
  - Information-theoretic: cannot reuse μ ☹

App: Bounded Storage Model



- Idea [DKRS]: authenticate w instead of S !!!
  - send $MAC_\mu(w)$ instead of $MAC_\mu(S)$
- Why does this help?
- Because W has high entropy for Eve !
  - "extractor-MAC": $MAC_\mu(W) \approx$ random
  - OK to reuse μ (if can build extractor-MACs) !!

Extractor-MACs

- Strong Extractor: $(I, \mathrm{Ext}(X, I)) \approx_\varepsilon (U_d, U_m)$ if X has min-entropy at least k
  - Goal 1: minimize d (note: $opt = \mathrm{O}(\log n + \log(1/\varepsilon))$),
  - Goal 2: maximize m (note: $opt = k - 2\log(1/\varepsilon) - \mathrm{O}(1)$)
- (Strong) One-time MAC: for any $x \neq x', y, y'$

$$\Pr_I(\mathrm{Ext}(x', I) = y' \mid \mathrm{Ext}(x, I) = y)| \leq \delta$$

  - Goal 1: minimize d (note: $opt = \mathrm{O}(\log n + \log(1/\delta))$),
  - Goal 2: minimize m (note: $opt = \log(1/\delta) + \mathrm{O}(1)$)
- Together: Extractor-MAC
  - Goals 1 & 2: minimize d, minimize m (MAC "wins")
  - Goal 3: minimize k (since want small m)

Extractor-MACs

- <u>Strong Extractor</u>: $(I, \mathrm{Ext}(X, I)) \approx_\varepsilon (U_d, U_m)$ if X has min-entropy at least k
  - <u>Goal 1</u>: minimize d (note: $opt = \mathrm{O}(\log n + \log(1/\varepsilon))$),
  - <u>Goal 2</u>: maximize m (note: $opt = k - 2\log(1/\varepsilon) - \mathrm{O}(1)$)
- <u>(Strong) One-time MAC</u>: for any $x \neq x', y, y'$

$$\Pr_I(\,\mathrm{Ext}(x', I) = y' \mid \mathrm{Ext}(x, I) = y) \mid \leq \delta$$

  - <u>Goal 1</u>: minimize d (note: $opt = \mathrm{O}(\log n + \log(1/\delta))$),
  - <u>Goal 2</u>: minimize m (note: $opt = \log(1/\delta) + \mathrm{O}(1)$)
- <u>Together</u>: Extractor-MAC. We achieve *optimal*
  - $d = \mathrm{O}(\log n + \log(1/\delta) + \log(1/\varepsilon))$, $m = \log(1/\delta) + \mathrm{O}(1)$,

    if $k \geq m + 2\log(1/\varepsilon) + \mathrm{O}(1) = \log(1/\delta) + 2\log(1/\varepsilon) + \mathrm{O}(1)$

Extractor-MACs

- Idea 1: pairwise independent hash functions are both extractors (universality) and one-time MACs
  - Optimal $m = \log(1/\delta)$ ☺, but long $d = n + \log(1/\delta)$ ☹
- Idea 2: compose with "almost universal" hash function before pairwise independence
  - Extractor part: OK if collision probability $\le 2^{-m}\varepsilon^2$ (so total $\le 2^{-m}(1+\varepsilon^2)$ and can still apply LHL),
  - MAC part: OK since pairwise independent MAC composes well with universal hash
- Optimize parameters to get the result

Robust Sketches & Extractors

- If the user can store only biometric w, how can he be sure that P or $S(w)$ are correct [BDKOS] ?
  - Robust Secure Sketches / Fuzzy Extractors
  - Server can only refuse to help or give correct $P/S(w)$
  - Applications to biometric authenticated key-exchange secure against man-in-the-middle attacks
- Idea: add "authentication information" H(pub,w) to the public information pub, for a special H
  - most work: finding H that works w/o leaking much info

Robust Sketches & Extractors

- Which H(pub,w) will produce a good MAC?
- [BDK$^+$05]:
  - H = Random Oracle. Works (still tricky)
- [DKRS06]: recall, pub=(S(w),h)
  - Use "interconnected" extractor h and MAC H
  - Works only if $k \geq n/2$ (inherent in this model ☹)
  - Extract (much) less than in "non-robust" case ☹
- [CDF$^+$08]: regain optimality using a CRS!
  - Idea: set pub=S(w), CRS = h and … more tricks

Concluding

- Randomness extractors are useful for
  - Key derivation
  - Privacy (entropic security!)
  - Many Combinations
- In many cases plain extractors not enough
  - Need "special-purpose" extractors

Special Purpose Extractors

- Adding Invertibility:
  - Entropically-Secure Encryption
- Adding Collision-Resistance:
  - Perfect one-way hash functions (POWHF)
- Adding Error-Correction:
  - Fuzzy extractors (FE), secure sketches (SS)
- Correcting errors w/o leaking partial info
  - Private FEs and SSs, fuzzy POWHFs
  - Error-correction in the bounded storage model
- Adding Authentication, Local Computability…

Concluding

- Randomness extractors are useful for
  - Key derivation
  - Privacy (entropic security!)
  - Many Combinations
- In many cases plain extractors not enough
  - Need "special-purpose" extractors
- Variants of leftover hash lemma very useful
- Unexpected tools, connections, subtleties
- Elegant techniques, nice insights
- Exciting area, many open questions left !!!


12
1
2
3
4
5
6
8
9
10
11