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: $\Pr[X=x] \leq 2^{-k}$, for all $x$

• Can we extract (nearly) perfect randomness from such realistic, imperfect sources?
Extractors: 1st attempt

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

- Impossible! $\exists$ set of $2^{n-1}$ inputs $x$ on which first bit of $\text{Ext}(x)$ is constant $\Rightarrow$ “flat” $(n-1)$-source $X$, bad for $\text{Ext}$. 
Modern Extractors [NZ]

- **Def:** 
  
  A $(k,\varepsilon)$-extractor is $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ 
  
  s.t. $\forall k$-source $X$, $\text{Ext}(X, U_d)$ is $\varepsilon$-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 $\text{Ext}(X, I)$ as the “effective” secret key.

• Def: $\text{Ext}$ is a $(k, \varepsilon)$ strong extractor if

\[
\text{Ext}'(x; i) = i \circ \text{Ext}(x, i) \text{ is a } (k, \varepsilon)\text{-extractor}
\]

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

• Universal Hash Family \( \{ h_i : \{0,1\}^n \rightarrow \{0,1\}^m \} \):

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

• Leftover Hash Lemma [HILL]: universal hash functions \( \{h\} \) yield strong extractors:

\( (I, h_I(X)) \approx_\epsilon (I, U_m) \)

- optimal output length: \( m = k - 2 \log(1/\epsilon) \)
- seed length: \( d = O(n) \)

• Ex: \( \text{Ext}(x;a) = \text{first } m \text{ bits of } a \cdot x \text{ in } GF(2^n) \)

• 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 $(k,\varepsilon)$-entropically secure if $\forall k$-source $X$, $\forall$ predictors, $\exists$ simulator, $\forall$ functions $f$, seeing $S(X)$ “does not help”:

$$\Pr\left(S(X) \xrightarrow{\text{Predictor}} f(X)\right) \leq \Pr\left(S(X) \xrightarrow{\text{Simulator}} f(X)\right) + \varepsilon$$

- Also say $S()$ hides all functions of $X$
- Notice, $S()$ must be probabilistic ($f = S$)
- $S()$ must also be one-way ($f = \text{identity}$)
- 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 $(k, \varepsilon)$-indistinguishable if
  \[ \forall k \text{-sources } X, Y, S(X) \text{ is } \varepsilon\text{-close to } S(Y) \]
  - In particular, all of them are $\varepsilon$-close to $S(U)$
  - $(k, \varepsilon)$-extractors are also $(k, 2\varepsilon)$-indistinguishable

• **Thm [DS1]:** If $S()$ is $(k, \varepsilon)$-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 $(k, \varepsilon)$-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 $(k, \varepsilon)$-extractor whose source can be recovered from its output and its seed.
Invertible Extractors

• If \( C = E(M; K) \), then we want
  1. \( C \approx \text{random} \), 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 $E(M,K) = N(M,K)$
  - Random step from $M$
  - Decryption assumes labeling is "invertible", which is easy to get (Cayley graphs)
- **Goal**: 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 $\delta$-biased if $|\text{bias}_\alpha(X)| \leq \delta$ for all $\alpha \neq 0$
- Can sample $\delta$-biased $X$ with $2\log(n/\delta)$ bits

- Fact: If $X$ is $\delta$-biased, $M$ is $k$-source then $M \oplus X \approx_\varepsilon \text{uniform}$, where $\varepsilon = \delta \cdot 2^{(n-k)/2}$
- Use optimal $\delta$-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 $\{h_i : \{0,1\}^d \rightarrow \{0,1\}^n\}$ 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
  
  $$E(I, M \oplus h_I(K)) \approx_\varepsilon E(I, U_n)$$

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

- **Theorem [DS1]:** three constructions
  - From expander graphs, achieve optimal $d = n - k + 2 \log(1/\epsilon)$, where $\epsilon$ is the “error”
  - “Sparse One-time Pad: $E(M; K) = M \oplus S(K)$, where $d = n - k + 2 \log(n/\epsilon)$
    - $S(K)$ is a point sampled from $(\epsilon \cdot 2^{(k-n)/2})$-biased set
  - “Probabilistic OTP”: get $d = n - k + 2\log(1/\epsilon)$
    - $E(M; K) = (I, M \oplus h_I(K))$
    - probabilistic encryption ($I$ is not part of $K$)
    - Here $\{h_i : \{0,1\}^d \rightarrow \{0,1\}^n\}$ is “XOR-universal”
Adding Collision-Resistance: Perfectly One-Way Hash Functions
Collision-Resistant Extractors

• Collision: \((w,i) \neq (w',i')\) s.t. \(\text{Ext}(w;i) = \text{Ext}(w';i')\)
  - Strong extractors: \(i, w \neq w'\) s.t. \(\text{Ext}(w;i) = \text{Ext}(w';i)\)

• “Commit” to \(w\) by publishing \((i, \text{Ext}(w;i))\)
  - Great decommitment: simply present \(w\)!

• **Entropic Security**: if entropy of \(W\) is at least \(k\), then \((I, \text{Ext}(W;I))\) hides all functions of \(W\) (weaker than usual hiding)

• Note: don’t need full power of extractors, suffices to have \((k,\varepsilon)\)-indistinguishability
Construction

• Yet another variant of LHL:
  - \( \text{Ext}(W; I) = f(h_I(W)) \)
  - \( f : \{0,1\}^N \rightarrow \{0,1\}^m \) 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 
  \[ \left( I, f(h_I(W)) \right) \approx_\varepsilon \left( I, f(U_N) \right) \]

• Apply with \( f = \text{CRHF} \) and family of pairwise independent permutations (e.g., \( \{ax+b|a\neq0\} \))
  - Permutations ensure collision-resistance

• **Gives** Perfectly One-Way Hash Functions and Obfuscators for Equality for inputs with entropy > output of CRHF + 2\log(1/\varepsilon)
Adding Locally Computable Aspect:
Key Derivation in Bounded Storage Model
Bounded Storage Model [Mau]

• **Setting:**
  - 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 $\gamma N$, 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 $i_1, \ldots, i_t \in \{1\ldots N\}$
  - $w = X[i_1] \ldots X[i_t]$, 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])

$R = \text{"Ext}(w,K)"$
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 $\text{Ext}(W; I)$ 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)

\[
\begin{array}{c}
w \\
\text{seed}
\end{array} \quad \xrightarrow{\text{Gen}} \quad \begin{array}{c}
R \\
P
\end{array}
\]

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

\[
\begin{array}{c}
w' \\
P
\end{array} \quad \xrightarrow{\text{Rep}} \quad R
\]

- \( R \) is nearly uniform given \( P \) if \( w \) has sufficient min-entropy (can put usual \( n, m, k, t, \varepsilon \))

- **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** $S(w)$

  $$w \xrightarrow{} S \xrightarrow{} S(w)$$

- **Recover** $w$ from $S(w)$ and **any** $w' \approx w$ ($w'$ close to $w$)

  $$w' \xrightarrow{} \text{Rec} \xrightarrow{} w$$

  $$S(w) \xrightarrow{} \text{Rec} \xrightarrow{} w$$

- $w$ 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)

- **Punchline**: 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

\[ S(w) = \text{syndrome}(w) \quad \text{OR} \quad S(w;r) = w \oplus \text{ECC}(r) \]

- 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 $P = (S(w), I)$, $R = \text{Ext}(w; I)$
  - Extract $|R| \approx \text{residual min-entropy} - 2\log(1/\varepsilon)$

Information-Reconciliation

Can we design sketches which leak no "useful" information about $w$?

But I still learned $S(w)$!

Still high uncertainty about $w$.

I think we can !!!
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 $(k, \varepsilon)$-entropically secure

• Notice, implies residual entropy $\geq \log(1/\varepsilon)$

• Converse false: code-offset leaked $\text{syn}(w)$

• Suffices to construct $(k, \varepsilon)$-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!

---

**Diagram:**
- $w$ is input
- Random coins
- $S$ is extractor
- $S(w)$ is output
- Recover
- $\approx_{\varepsilon}$ uniform
- $\exists$ (strong) extractor $S(\cdot)$ (for Hamming errors)
Error-Correcting Extractors

• Idea 1: Recall, \( S(W; X) = W \oplus X \approx \epsilon \) uniform, if \( X \) is \( \epsilon^2 \) biased

• Idea 2: Recall, \( S(W; X) = W \oplus X \) is a good sketch if \( X \) is a random codeword in a good code

• Can we achieve both simultaneously?
  - Yes for non-linear codes, but no explicit constructs 😞
  - No for linear codes (any \( \alpha \) in the dual has \( \alpha \odot X = 0 \)) 😞

• Idea 3: use a family of (carefully chosen) linear codes to get the best of both worlds!

Recently constructed by Shpilka’05 (bad params though)
Construction

• Design family of codes \{\text{ECC}_i\} and set

\[ S(w; i) = (i, \text{syn}_i(w)) \quad \text{OR} \quad S(w; i, r) = (i, w \oplus \text{ECC}_i(r)) \]

\[ w \quad \downarrow \]

\[ i \rightarrow S \]

\[ i \]

\[ \text{syn}_i(w) \approx \varepsilon \]

\[ \text{uniform} \]

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

• for “large” alphabets get optimal parameters!
Construction

• Design family of codes \{ECC_i\} and set

\[ S(w;i) = (i, \text{syn}_i(w)) \text{ OR } S(w;i,r) = (i, w \oplus ECC_i(r)) \]

• **Theorem [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 ⇒ fuzzy extractor: set $P = (S(w), I)$, $R = \text{Ext}(w; I)$
  
  - Let’s use “extractor-sketches” instead!

• Get FE where $(P, R) \approx_{\varepsilon} (U_1, U_2)$
  
  - Even joint pair $(P, R)$ hides all functions of $W$!

• Called Private Fuzzy Extractors:
  
  - As opposed to usual fuzzy extractors, public data $P$ does not reveal anything “useful” about the biometric $W$, even if the key $R$ is leaked!
• Recall, POWHFs allow to publish a value $Z = \text{"Commit}(w)\"$ s.t. given input $w'$
  – Verify($Z,w'$) accepts if and only if $w = w'$
  – Moreover, $Z$ is $(k,\varepsilon)$-entropically secure

• What if want to test if $\text{distance}(w,w') < t$ ?

• Attempt: use secure sketch and publish $(Z, S(w))$
  – 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

- Shared secret sampling key $sk$
- Goal: $\text{H}(W_{sk} | S(W_{sk}), sk)$, $\text{H}(|S(W_{sk})|, sk)$ for Eve
- "Everlasting security": can we re-use $sk$?
- [Ding]: Not with usual sketches!
  - $S(W_{sk})$ leaks info on $sk$
- Extracting sketch: $S(W_{sk1}) \approx S(W_{sk2})$!
Adding Authentication: entropically-secure MACs, Robust FE/SS, ...
App: Bounded Storage Model

- Need to authenticate $S$
- No problem: add MAC key $\mu$ to $sk$
  - send $\text{MAC}_\mu(S)$ together with $S$
- But which MAC???
  - Computational: lose information-theoretic security 😞
  - Information-theoretic: cannot reuse $\mu$ 😞
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 \( \mu \) (if can build extractor-MACs) !!
Extractor-MACs

• **Strong Extractor**: \((I, \text{Ext}(X, I)) \approx_{\varepsilon} (U_d, U_m)\) if \(X\) has min-entropy at least \(k\)
  - **Goal 1**: minimize \(d\) (note: \(\text{opt} = O(\log n + \log(1/\varepsilon))\)),
  - **Goal 2**: maximize \(m\) (note: \(\text{opt} = k - 2\log(1/\varepsilon) - O(1)\))

• **(Strong) One-time MAC**: for any \(x \neq x', y, y'\)
  \[
  \Pr_{I}(\text{Ext}(x', I) = y' \mid \text{Ext}(x, I) = y) \leq \delta
  \]
  - **Goal 1**: minimize \(d\) (note: \(\text{opt} = O(\log n + \log(1/\delta))\)),
  - **Goal 2**: minimize \(m\) (note: \(\text{opt} = \log(1/\delta) + O(1)\))

• **Together**: Extractor-MAC
  - **Goals 1 & 2**: minimize \(d, m\) (MAC “wins”)
  - **Goal 3**: minimize \(k\) (since want small \(m\))
Extractor-MACs

- **Strong Extractor:** \((I, \text{Ext}(X, I)) \approx_\epsilon (U_d, U_m)\) if \(X\) has min-entropy at least \(k\)
  - **Goal 1:** minimize \(d\) (note: \(opt = O(\log n + \log(1/\epsilon))\)),
  - **Goal 2:** maximize \(m\) (note: \(opt = k - 2\log(1/\epsilon) - O(1)\))

- **(Strong) One-time MAC:** for any \(x \neq x', y, y'\)
  \[\Pr_I(\text{Ext}(x', I) = y' | \text{Ext}(x, I) = y) \leq \delta\]
  - **Goal 1:** minimize \(d\) (note: \(opt = O(\log n + \log(1/\delta))\)),
  - **Goal 2:** minimize \(m\) (note: \(opt = \log(1/\delta) + O(1)\))

- **Together:** **Extractor-MAC.** We achieve optimal
  
  \[d = O(\log n + \log(1/\delta) + \log(1/\epsilon)), m = \log(1/\delta) + O(1),\]

  if \(k \geq m + 2\log(1/\epsilon) + O(1) = \log(1/\delta) + 2\log(1/\epsilon) + O(1)\)
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 $\leq 2^{-m}\varepsilon^2$ (so total $\leq 2^{-m}(1+\varepsilon^2)$ 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 $H(\text{pub},w)$ will produce a good MAC?
- $[\text{BDK}^+05]$:
  - $H =$ Random Oracle. Works (still tricky)
- $[\text{DKRS06}]$: recall, $\text{pub}=(S(w),h)$
  - 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 😞
- $[\text{CDF}^+08]$: regain optimality using a CRS!
  - Idea: set $\text{pub}=S(w)$, $\text{CRS} = h$ 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 !!!