The spammed code offset method

The Spammed Code Offset Method

Boris Škorić and Niels de Vreede

Abstract

Helper data schemes are a security primitive used for privacy-preserving biometric databases and Physical Unclonable Functions. One of the oldest known helper data schemes is the Code Offset Method (COM). We propose an extension of the COM: the helper data is accompanied by many instances of fake helper data that are drawn from the same distribution as the real one. While the adversary has no way to distinguish between them, the legitimate party has more information and can *see the difference. We use an LDPC code in order to improve the efficiency of the legitimate party's selection procedure.*
Our construction provides a new kind of trade-off: more effective use of the source entropy, at the price of increased helper data storage. We give a security analysis in terms of Shannon entropy and order-2 Rényi entropy. We also propose a variant of our scheme in which the helper data list is not stored but pseudorandomly generated, changing the trade-off to source entropy utilization vs. computation effort.

1 Introduction

1.1 Helper Data Systems

The past decade has seen a lot of interest in a field that can be characterized as 'security with noisy data'. In several security applications it is necessary to *reproducibly* extract secret data from *noisy* measurements on a physical system.
One such application is the privacy-preserving storage of biometric data. Analogously to password hashing, one can store biometric data in hashed form in order to prevent inside attackers from learning what the enrolled biometric features look like. Another application is read-proof storage of cryptographic keys using Physical Unclonable Functions (PUFs) [28, 29, 26, 6, 22].
Storage of keys in nonvolatile digital memory can often be considered insecure because of the vulnerability to physical attacks. (For instance, fuses can be optically inspected with a microscope; flash memory may be removed and read out.) PUFs provide an alternative way to store keys, namely in *analog* form, which allows the designer to exploit the inscrutability of analog physical behavior. Keys stored in this way are sometimes referred to as Physically Obfuscated Keys (POKs) [16].
In both the biometrics and the PUF/POK application, one faces the problem that some form of error correction has to be performed, but under the constraint that the redundancy data, which is considered to be visible to attackers, does not endanger the secret extracted from the physical measurement. This problem is solved by a special security primitive, the *Helper Data System* (HDS).
A HDS in its most general form is shown in Fig. 1. The `Enroll` procedure takes as arguments a measurement X and (optionally) a random value R. It outputs a secret S and Helper Data W. The helper data is stored. In the reconstruction phase, a fresh measurement X' is obtained. Typically X' is a noisy version of X, close to X (in terms of e.g. Euclidean distance or Hamming distance) but not necessarily identical. The `Rec` (reconstruction) procedure takes X' and W as input. It outputs $\hat{S}$, an estimate of S. If X' is not too noisy then $\hat{S} = S$.
Two special cases of the general HDS are the Secure Sketch (SS) and the Fuzzy Extractor (FE) [12].

- The Secure Sketch has $S = X$ (and $\hat{S} = \hat{X}$, an estimator for X). If X is not uniformly distributed, then S is not uniform. The SS is suitable for privacy-preserving biometrics, where high entropy of S (given W) is required, but not uniformity.

- The Fuzzy Extractor has a (nearly) uniform S given W. The FE is typically used for extracting keys from PUFs and POKs.

There exists a generic construction to create a FE out of a SS: hashing the output of the SS using a Universal Hash Function (UHF) [8, 27, 20].

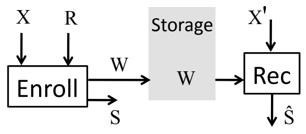

Figure 1: *Data flow in a generic Helper Data System.*

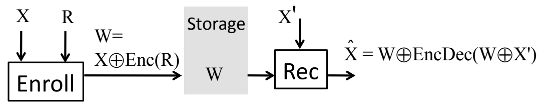

Figure 2: *The Code Offset Method employed as a Secure Sketch.*

1.2 The Code Offset Method

One of the oldest known SS constructions is the Code Offset Method (COM) [19, 12]. Here X is a binary string, say of length n, with probability distribution ρ. The construction uses a linear error-correcting code that encodes k-bit messages as n-bit codewords. The encoding and decoding operations are denoted as `Enc` and `Dec` respectively. The COM takes R uniformly drawn from $\{0,1\}^k$ and

$$W = X \oplus \mathtt{Enc}(R) \;\; ; \;\; \hat{S} = \hat{X} = W \oplus \mathtt{Enc}(\mathtt{Dec}(W \oplus X')). \quad (1)$$

This is depicted in Fig. 2. If X is uniformly distributed on $\{0,1\}^n$ then the scheme is not only a SS but in fact also a FE; this holds because a uniform X gives rise to helper data

W that leaks nothing about R. The formulas for the FE are: $S = R$ and $W = X \oplus \texttt{Enc}(R)$; $\hat{S} = \hat{R} = \texttt{Dec}(X' \oplus W)$.
Remark: The COM would work equally well if W were to point from X to the codeword closest to X instead of pointing to a random codeword. However, finding the nearest codeword for an arbitrary string is generally a difficult problem. Thus, the role of the auxiliary variable R in the COM is merely to circumvent this problem.
Note that for uniform X the W reveals the syndrome of X, but nothing about R. Hence R can then be used as a cryptographic key. In this paper we study *non-uniform* X. A non-uniform X appears naturally, e.g. in the case of Coating PUFs [28], where Gray-coded capacitance measurements are concatenated to form X. Typically not all Gray code words are represented, which leads to non-uniformity.
Similarly, a biometric feature vector is often split up into near-independent components which each yield a small number of non-uniform bits.

1.3 Zero Leakage

For some sources X it is possible to define helper data that reveals *nothing* about S. This is sometimes called Zero Secrecy Leakage (ZSL) helper data. The information contained in X is split into two *independent* parts, one of which serves for error correction, and one for constructing the secret S.
As we saw above, the COM with uniform X has the ZSL property. Another example is the quantile partitioning scheme [31] of Verbitskiy et al. for *continuous* X, and its generalization to non-uniform S [10].

1.4 Contributions and outline

In this paper we propose a simple modification of the Code Offset Method Secure Sketch. The basic idea is to hide the helper data amid a number (say $m - 1$) of dummy helper data instances. All the instances are a priori indistinguishable from the point of view of the adversary, The legitimate party, on the other hand, possesses X', which allows for efficiently finding the correct helper data instance. This workload asymmetry improves the security. For small m, the attacker may simply try out all possibilities, which leads to an average attack effort of $(m + 1)/2$ times the original effort. For very large m this brute force attack is no longer feasible, and the attacker is forced to ignore the public data; in this way a new kind of 'zero leakage' is achieved, distinct from the ZSL of Section 1.3, namely public data that reveals practically nothing about X (as opposed to zero leakage about S).
The concept of 'spamming' the attacker in this way is very general and is applicable whenever there exists an efficient way of recognizing the correct W using X'. In this paper we show how the 'spamming' concept can be applied to the Code Offset Method. Our scheme requires the use of a linear error-correcting code with low-density parity check matrix (LDPC) in order to keep the legitimate party's workload low.
In Section 2 we introduce notation and assumptions. In Section 3 we discuss a version of the COM that does not need the auxiliary variable R. We call it the Syndrome-Only COM. We analyze its leakage and briefly review the Leftover Hash Lemma. In Section 4 we present an information-theoretic analysis of the generic principle of adding fake data. In Section 5 we present our new scheme, which we call the Spammed Code Offset Method (SCOM). Section 6 contains a security analysis of the SCOM. Memory requirements and search efficiency are discussed in Section 7. In Section 8 we present a modified version of the SCOM where all the helper data is generated by a pseudo-random number generator instead of being retrieved from storage. We call this version GSCOM (Generative SCOM). A discussion and conclusions are given in Section 9.

2 Notation and attacker model

Random variables are written with capitals, and their realizations in lower case. Vectors are in boldface; sets in calligraphic font. Concatenation is denoted as $||$. The notation $d_{\mathrm{H}}(x, y)$ stands for the Hamming distance between x and y. The logarithm 'log' has base 2. The natural logarithm is ln. The unit vector $\boldsymbol{e}_j$ consists of all zeros, except for a '1' in position j. The Kronecker delta is written as $\delta_{x,y}$.
The Code Offset Method works with a *linear* code $\mathcal{C}$ that has n-bit code words and k-bit messages. The encoding and decoding algorithms associated with this code are denoted as $\texttt{Enc}\colon \{0,1\}^k \to \{0,1\}^n$ and $\texttt{Dec}\colon \{0,1\}^n \to \{0,1\}^k$ respectively. The algorithm for computing the syndrome is denoted as $\texttt{Syn}\colon \{0,1\}^n \to \{0,1\}^{n-k}$. If the decoder is a syndrome decoder, then the mapping from a syndrome to a minimal error pattern is called $\texttt{SDec}\colon \{0,1\}^{n-k} \to \{0,1\}^n$.
We consider a source (biometric/POK) whose output at enrollment is a bit string $X \in \{0,1\}^n$. The string R in Fig. 2 has length k. The helper data is called W. In the FE setting, the cryptographic key that is ultimately derived is denoted as $Q \in \{0,1\}^\ell$.
We will use shorthand notation $p_{xw} = \Pr[X = x, W = w]$, $p_w = \Pr[W = w]$ and $p_{x|w} = \Pr[X = x|W = w]$, when it does not cause ambiguity. We define $q_z = \Pr[\texttt{Syn}(X) = z]$. The public data stored in nonvolatile memory is P.
The outcome of the measurement in the reconstruction phase is denoted as $X' \in \{0,1\}^n$. The X' is a noisy version of X, and in general does not have the same probability distribution as X. The estimator for the key Q, derived from X' and the public data, is denoted as $\hat{Q}$.
We will rely on a cryptographic hash function f. Furthermore we will use a Universal Hash Function $g(x, a)$, where the second argument is public auxiliary randomness.
The Shannon entropy of a random variable X is written as $\mathsf{H}(X)$. Conditional Shannon entropy is denoted as $\mathsf{H}(X|Y)$. The mutual information between X and Y is written as $I(X;Y)$. The Kullback-Leibler distance from X to Y is $D(X||Y)$.
The attacker model is summarized as follows. We distinguish between two scenarios:

1. <u>Biometric database for authentication</u>.
   The adversary can read but not manipulate the public data P. His aim is to learn as much about X as he can.[1]

2. <u>Secure key storage with a POK</u>.
   The adversary has access to the device which contains the POK. He cannot re-activate the device's enrollment mode of operation. The opacity of the POK, and the embedding of the POK in the device, prevent the adversary

[1] He may exploit this knowledge in various ways: (i) Some part of X may reveal information about medical conditions. This is a privacy risk. (ii) Construct a fake biometric in order to pass authentication. This is a security risk. (iii) Cross-linking of people across different databases. This is a privacy risk.

from reading out Q from the POK. Furthermore, physical tampering with the POK is unerringly detected by the device at the reconstruction phase. The public data P is stored on the device in insecure nonvolatile memory. The adversary is able to read and to manipulate P. There is no Public Key Infrastructure that would allow the device to verify the authenticity of the public data. The adversary's main aim is to learn the POK key Q. A secondary goal is to cause the device to accept a key other than Q as the correct key.

In both scenarios the adversary is able to discern whether reconstruction is successful. He also observes the running time of the reconstruction algorithm. We assume that no other side channels exist.

3 The Syndrome-Only COM

3.1 Construction

As was mentioned in Section 1, the only role of the auxiliary variable R in the COM is to avoid the difficult problem of finding the codeword closest to X. Here we present an alternative way of avoiding this problem. This construction does not need auxiliary random variables, and thereby significantly simplifies security analyses (see Sections 3.2, 4 and 6). We present two versions: (1) A version that requires a syndrome decoder `SDec`. See e.g. construction 3 in [11]. (2) A version without such a requirement. Instead, it makes use of a lookup table that allows arbitrary syndromes to be mapped to n-bit words in a deterministic way.
Version 2 needs a separate Setup phase. The two versions have the same enrollment algorithm, but their reconstruction algorithms differ. We start by presenting version 1. (The abbreviation SO stands for 'Syndrome-Only'.)

Algorithm SO1.Enroll

```
1. Measure $X \in \{0,1\}^n$.
2. Compute helper data $W = \mathtt{Syn}\, X$.
3. Generate salt $\zeta$. Compute $h = f(W||\zeta||X)$.
4. Publicly store $P = (W, \zeta, h)$.
```

Algorithm SO1.Reconstruct

```
1. Read $P' = (W', \zeta', h')$.
2. Measure $X' \in \{0,1\}^n$.
3. Compute $\sigma = W' \oplus \mathtt{Syn}\, X'$.
4. $E = \mathtt{SDec}\, \sigma$.
5. $\hat{X} = X' \oplus E$.
6. $\hat{h} = f(W'||\zeta'||\hat{X})$.
7. Abort with failure if $\hat{h} \neq h'$.
```

We note the following,

- The size of the helper data W is only $n-k$ bits, as compared to the n bits in the original COM.
- Steps 3 and 4 of SO1.Reconstruct make use of the linearity of the syndrome: $\sigma = W' \oplus \mathtt{Syn}\, X' = \mathtt{Syn}\, X \oplus \mathtt{Syn}\, X' = \mathtt{Syn}(X' \oplus X)$. This makes it possible to perform the steps prior to decoding without leaving the 'syndrome space' $\{0,1\}^{n-k}$.
- This construction is compatible with the 'reverse fuzzy extractor' protocol proposed in [30], and similar protocols where the decoding step is outsourced.
- At reconstruction the public data may have been (maliciously) altered, which is why we use the notation P', W', ζ', h' in step 1 of algorithm SO1.Reconstruct.
- In step 3 of SO1.Enroll, X is hashed *together with the helper data*. This way of protecting the helper data against manipulation was introduced by [7]. Alternatively, one may use a Message Authentication Code with Key Manipulation Security [9].

Algorithm SO2.Setup

```
1. For $j \in \{1, \ldots, n-k\}$:
   Find $L_j \in \{0,1\}^n$ such that $\mathtt{Syn}\, L_j = \boldsymbol{e}_j$.
2. Publicly store $L = (L_j)_{j=1}^{n-k}$.
```

SO2.Enroll = SO1.Enroll.

Algorithm SO2.Reconstruct

```
1. Read $P' = (W', \zeta', h')$.
2. Measure $X' \in \{0,1\}^n$.
3. Compute $V = X' \oplus \sum_{j \in \{1,\ldots,n-k\}} W'_j L_j$.
4. $E = V \oplus \mathtt{Enc}(\mathtt{Dec}(V))$.
5. $\hat{X} = X' \oplus E$.
6. $\hat{h} = f(W'||\zeta'||\hat{X})$.
7. Abort with failure if $\hat{h} \neq h'$.
```

- In SO2.Setup, finding values L_j is not difficult. It requires having the pseudo-inverse of the parity check matrix.
- In SO2.Setup, the choice of L_j is arbitrary, in the same way that the R in the original COM is arbitrary.
- The size of the table L is merely $(n-k) \times n$ bits.
- In step 3 of SO2.Reconstruct, the sum $\sum_j$ is understood to be defined as bitwise XOR.
- L is public. Hence, in case SO2.Reconstruct is implemented on a severely resource-constrained device that cannot store L, the table lookup and the computation of $\sum_j W'_j L_j$ can be outsourced.
- The V in step 3 of SO2.Reconstruct is the analogue of $X' \oplus W$ in the ordinary COM. The V contains a lot of information about X, so it must not be revealed to other parties. The decoding of V can be outsourced only if V is masked first.
- The `Dec` can be *any* decoder algorithm.
- An alternative (and security-wise equivalent) enrollment procedure would be to store $\sum_j (\mathtt{Syn}\, X)_j L_j \in \{0,1\}^n$ instead of $\mathtt{Syn}\, X$ as helper data. That would constitute a *deterministic* version of the helper data $X \oplus \mathtt{Enc}\, R$ in the original COM. All the random choices have been shifted to the creation of the lookup table in the setup phase, where they are 'frozen' as system parameters.

3.2 Analysis of the Syndrome-only COM

The lack of auxiliary variables makes our scheme simpler to analyze than the original COM. We consider the general case of a source variable X that is not necessarily uniform.

As mentioned in Section 2, the probability $\Pr[X = x]$ is denoted as p_x and $\Pr[\mathtt{Syn}\,X = y]$ as q_y. Thus we have $\Pr[W = w] = q_w$.

Lemma 1 *In the Syndrome-Only COM we have the following probabilities:*

$$p_{w|x} = \delta_{w,\mathtt{Syn}\,x} \quad p_{xw} = p_x \delta_{w,\mathtt{Syn}\,x} \quad p_{x|w} = \frac{p_x \delta_{w,\mathtt{Syn}\,x}}{q_w}.$$

Proof: The $p_{w|x}$ follows trivially from the fact that $W = \mathtt{Syn}\,X$. Multiplication by p_x yields p_{xw}. Finally $p_{x|w}$ is equal to p_{xw}/q_w. □

Lemma 2 *In the Syndrome-Only COM it holds that*

$$I(W;X) = \mathsf{H}(\mathtt{Syn}\,X), \qquad \mathsf{H}(W|X) = 0$$
$$\mathsf{H}(X|W) = \mathsf{H}(X) - \mathsf{H}(\mathtt{Syn}\,X).$$

Proof: The first two equations immediately follow from $W = \mathtt{Syn}\,X$. In the last line we write $\mathsf{H}(X|W) = \mathsf{H}(X,W) - \mathsf{H}(W)$ with $\mathsf{H}(X,W) = \mathsf{H}(X)$ and $\mathsf{H}(W) = \mathsf{H}(\mathtt{Syn}\,X)$. □

We briefly discuss the required amount of compression in case one wants to build a Fuzzy Extractor from the Secure Sketch. In order to obtain a nearly uniform key Q from X, one has to hash down to a smaller size (say ℓ): $Q = g(X, A) \in \{0,1\}^\ell$. Here A is *public* auxiliary randomness that serves as a 'catalyst' for the UHF g.

Let U be a uniform variable on $\{0,1\}^\ell$. The relation between ℓ and the uniformity of Q is given by the Leftover Hash Lemma (LHL) [17] and can be formulated as

$$\ell \leq L_\varepsilon(X,W) \quad \Longrightarrow \quad \mathbb{E}_w[\Delta(U; Q|W = w)] \leq \varepsilon \qquad (2)$$

$$\text{with } L_\varepsilon(X,W) = \mathsf{H}_2(X|W) + 1 - 2\log\frac{1}{\varepsilon}. \qquad (3)$$

Eq. (2) states that the non-uniformity of Q given W does not exceed ε as long as X has been sufficiently hashed down. The ℓ must not exceed the 'ε-extractable randomness' L_ε. The notation H_2 in (3) stands for the conditional Rényi entropy of order two and is defined as [13]

$$\begin{aligned} \mathsf{H}_2(X|W) &= -2\log T_2(X|W) \\ T_2(X|W) &= \mathbb{E}_w \sqrt{\textstyle\sum_x p_{x|w}^2} = \sum_w \sqrt{\textstyle\sum_x p_{xw}^2}, \end{aligned} \qquad (4)$$

where $\mathbb{E}_w$ stands for the expectation value over W. Note that the 'penalty' term $2\log\frac{1}{\varepsilon}$ in (3) depends only on ε, i.e. it depends not on the *improvement* of the uniformity but on the *final* uniformity. Because of this fact, the approach using UHFs can be quite wasteful.

Remark: Under some conditions [4] the factor 2 in the penalty term can be replaced by 1. Furthermore, the LHL can be sharpened somewhat by considering *smooth* Rényi entropy [23, 24, 32]. Such details are beyond the scope of the current paper.

4 Adding fake helper data; general considerations

The concept of hiding data in a large amount of fake data is very old. However, the application of this principle to *helper data* is, to the best of our knowledge, new.

Lemma 3 *Let W be helper data. Let $W_1^{\rm fake}, \ldots, W_{m-1}^{\rm fake}$ be i.i.d. generated fake helper data instances (independent of W), where $m \geq 1$, and where it is not necessarily the case that the probability distribution of the fake helper data is the same as for the true helper data. Let $Z \in \{1,\ldots,m\}$ be a random variable, not necessarily uniform. Let $\mathbf{\Omega}$ be a vector constructed from the real and the fake helper data as $\mathbf{\Omega} = (W_1^{\rm fake}, \ldots, W_{Z-1}^{\rm fake}, W, W_Z^{\rm fake}, \ldots, W_{m-1}^{\rm fake})$. Then*

$$\mathsf{H}(X|\mathbf{\Omega}) = \mathsf{H}(X|W) + \mathsf{H}(W|\mathbf{\Omega}) - \mathsf{H}(W|X\mathbf{\Omega}). \qquad (5)$$

If W is a function of X only, then

$$\mathsf{H}(X|\mathbf{\Omega}) = \mathsf{H}(X|W) + \mathsf{H}(W|\mathbf{\Omega}). \qquad (6)$$

Proof: We apply the chain rule to the expression $\mathsf{H}(XW|\mathbf{\Omega})$ in two different ways and obtain $\mathsf{H}(X|\mathbf{\Omega}) + \mathsf{H}(W|X\mathbf{\Omega}) = \mathsf{H}(W|\mathbf{\Omega}) + \mathsf{H}(X|W\mathbf{\Omega})$. Since the fake helper data are independent of X and W we have $\mathsf{H}(X|W\mathbf{\Omega}) = \mathsf{H}(X|W)$. This proves (5). Finally, if W is a function of X only, then $\mathsf{H}(W|X) = 0$ and consequently $\mathsf{H}(W|X\mathbf{\Omega}) = 0$. □

Remark 1: In this lemma it is clearly advantageous for the analysis if W is computed from X without auxiliary randomness. In case algorithm SO2 is used, remember that the lookup table L is not a random variable during enrollment and verification, even if it was created randomly during Setup.

Remark 2: In (6) we see a nice separation into a term $\mathsf{H}(X|W)$, which is the 'old' result for an ordinary helper data scheme, and the contribution $\mathsf{H}(W|\mathbf{\Omega})$ that arises from hiding W in a random position in the list $\mathbf{\Omega}$.

The 'hiding' term $\mathsf{H}(W|\mathbf{\Omega})$ can be further broken down as follows.

Lemma 4 *Let W be a function of X only. Let Z and $\mathbf{\Omega}$ be defined as in Lemma 3. Then*

$$\mathsf{H}(W|\mathbf{\Omega}) = \mathsf{H}(Z) - \underbrace{\mathsf{H}(Z|W\mathbf{\Omega})}_{\substack{\text{collision}\\ \text{penalty}}} - \underbrace{I(Z;\mathbf{\Omega})}_{\substack{\text{distribution}\\ \text{mismatch penalty}}}. \qquad (7)$$

If Z is drawn uniformly, and the fake helper data have the same distribution as W, then

$$\mathsf{H}(W|\mathbf{\Omega}) = \log m - \mathsf{H}(Z|W\mathbf{\Omega}). \qquad (8)$$

Proof: Using the chain rule we have $\mathsf{H}(W|\Omega) = \mathsf{H}(WZ\mathbf{\Omega}) - \mathsf{H}(\mathbf{\Omega}) - \mathsf{H}(Z|W\mathbf{\Omega})$. Next we expand $\mathsf{H}(WZ\mathbf{\Omega})$ as $\mathsf{H}(Z) + \mathsf{H}(\mathbf{\Omega}|Z) + \mathsf{H}(W|Z\mathbf{\Omega})$, where $\mathsf{H}(W|Z\mathbf{\Omega}) = 0$. This yields $\mathsf{H}(W|\mathbf{\Omega}) = \mathsf{H}(Z) - \mathsf{H}(Z|W\mathbf{\Omega}) - [\mathsf{H}(\mathbf{\Omega}) - \mathsf{H}(\mathbf{\Omega}|Z)]$, which is precisely (7). Next, if the variables $W_j^{\rm fake}$ are distributed precisely like W, then $\mathbf{\Omega}$ reveals no information about Z, i.e. $I(Z;\mathbf{\Omega}) = 0$. Finally, if Z is uniform on $\{1,\ldots,m\}$ then $\mathsf{H}(Z) = \log m$. □

Note that there are two clearly interpretable penalty terms in (7). The 'collision penalty' $\mathsf{H}(Z|W\mathbf{\Omega})$ increases with m. It becomes non-negligible when $\mathbf{\Omega}$ contains so many entries that it becomes likely that there exist entries with the same value;[2] then even knowing W and $\mathbf{\Omega}$ does not fix Z.

The 'distribution mismatch penalty' occurs when the fake entries in $\mathbf{\Omega}$ do not look statistically the same as W; then some information about Z can be obtained already from inspecting $\mathbf{\Omega}$.

At first sight (6) might seem to contradict the well known principle 'conditioning reduces Shannon entropy'. However, it should be borne in mind that $\mathbf{\Omega}$ is not just W plus decoys; $\mathbf{\Omega}$ results from a Z-dependent *function* applied to W and the decoys. This function reduces the leakage from W. Lemmas 3 and 4 will be used for the analysis of the Spammed COM in Section 6.1.

Lemma 5 *Let the helper data W be a function of X only. Let Z and $\mathbf{\Omega}$ be defined as in Lemma 3. Let $W^{\rm fake}$ denote an arbitrary fake entry in $\mathbf{\Omega}$. Let $\pi_z = \Pr[Z = z]$. Then*

$$I(Z;\mathbf{\Omega}) \leq (1 - \sum_{z=1}^{m} \pi_z^2) \left[D(W||W^{\rm fake}) + D(W^{\rm fake}||W)\right].$$

Proof: See Appendix. □

Note: if Z is uniform then $\sum_z \pi_z^2 = 1/m$.

5 The Spammed Code Offset Method

We first show a naive spamming approach, without efficient de-spamming at the reconstruction phase. Then we propose an efficient scheme, in two variants: one in the privacy-preserving biometrics context, the other in the secure key storage context. The efficient scheme requires a linear block code with a *low-density* parity check (LDPC) matrix [15]. (For background on LDPC codes see e.g. [25, 14].)

We present our SCOM schemes as being derived from the scheme SO1, for reasons of simplicity and brevity. However, SO2 can be chosen as the underlying scheme instead. The security and the storage requirements are analyzed in Section 6.

5.1 Naive approach

```
Algorithm NaiveSCOM.Enroll
  1. Measure $X \in \{0,1\}^n$.
  2. Compute helper data $W = \texttt{Syn}\, X$.
  3. For $j \in \{1,\ldots,m-1\}$: draw $W_j^{\rm fake} \in \{0,1\}^{n-k}$ from
     the distribution of $\texttt{Syn}\, X$.
  4. Uniformly draw $Z \in \{1,\ldots,m\}$.
  5. Construct a vector
     $\mathbf{\Omega} = (W_1^{\rm fake}, \cdots, W_{z-1}^{\rm fake}, W, W_z^{\rm fake}, \cdots, W_{m-1}^{\rm fake})$.
  6. Compute $G = f(\mathbf{\Omega}||X)$.
  7. Store public data $P = (\mathbf{\Omega}, G)$.
```

[2]It is possible to avoid collisions during the construction of $\mathbf{\Omega}$. We will not consider this approach since it has very limited benefit.

```
Algorithm NaiveSCOM.Reconstruct
  1. Read $P' = (\mathbf{\Omega}', G')$.
  2. Measure $X' \in \{0,1\}^n$.
  3. Compute $\sigma = \texttt{Syn}\, X'$.
  4. Set $\mathcal{L}_1 = \emptyset$. For $j \in \{1,\ldots,m\}$ do:
     (a) Try to compute $E_j = \texttt{SDec}(\Omega'_j \oplus \sigma)$.
     (b) If the decoding succeeds then add $j$ to $\mathcal{L}_1$.
  5. If $\mathcal{L}_1 = \emptyset$ then abort with failure.
  6. Set $\mathcal{L}_2 = \emptyset$. For $i \in \mathcal{L}_1$ do:
     (a) $\hat{X}_i = X' \oplus E_i$.
     (b) Compute $G_i = f(\mathbf{\Omega}'||\hat{X}_i)$.
     (c) If $G_i = G'$ then add $i$ to the list $\mathcal{L}_2$.
  7. If $|\mathcal{L}_2| \neq 1$ then abort; else $\hat{X} = X_{\mathcal{L}_2}$.
```

It is crucial that the adversary cannot 'see inside' the hash G. Note that in step 6 of the enrollment, the hash is computed over the *entire* vector $\mathbf{\Omega}$. This ensures that any manipulation of the public data will be detected, be it in the hash, in W or in the decoys. Note that the fake helper data also play the role of salt.

In step 3 of the enrollment, an alternative would be to draw a fake $X_j^{\rm fake}$ from the distribution of X and then compute $W_j = \texttt{Syn}\, X_j^{\rm fake}$.

At reconstruction the public data may have been altered, which is why we use the notation P', $\mathbf{\Omega}'$, G' in step 1 of the reconstruction. A list $\mathcal{L}_1$ is made of $\mathbf{\Omega}'$ entries that lead to successful decoding. The whole set $\mathcal{L}_1$ has to be taken into account, since some of the decoys may by chance decode, and the order of the entries is random. The list of candidates is further narrowed down to a list $\mathcal{L}_2$ of entries whose $\hat{X}_j$ generates the correct hash. If $P' = P$ and $X' \approx X$ then typically there is only one candidate left in $\mathcal{L}_2$. If $P' \neq P$ or X' is too noisy to be error-corrected, then typically $\mathcal{L}_2 = \emptyset$.

The main idea behind the scheme is that the adversary cannot distinguish between the true helper data and the decoys. The legitimate party, on the other hand, knows X', which allows it to make the distinction.

It may happen that the choice of system parameters is such that NaiveSCOM.Reconstruct has a long running time. Step 4 contains m decodings, and $|\mathcal{L}_1|$ hashes are computed in step 6. The choice of n, k, and m may give rise to a long list $\mathcal{L}_1$. In the schemes below we aim to reduce the running time of the reconstruction algorithm.

Note that the running time may reveal the length of $\mathcal{L}_1$, and possibly of $\mathcal{L}_2$. We do not consider this to be a security leak, since the $|\mathcal{L}_1|$ and $|\mathcal{L}_2|$ reveal only the amount of noise in X'.

5.2 Secure Sketch for biometrics database

Below we show a more efficient pair of algorithms (BioSCOM.Enroll, BioSCOM.Verify) in the biometric verification scenario. The idea behind this scheme is that comparing $\texttt{Syn}\, X'$ to $\texttt{Syn}\, X$ and the other syndromes allows the device to heuristically re-order $\mathbf{\Omega}'$ in such a way that the most likely candidates are tried first. Here it is crucial that the parity check matrix of the code has low density: then a small Hamming distance between X and X' leads to a small Hamming distance between $\texttt{Syn}\, X$ and $\texttt{Syn}\, X'$.

Processing $\mathbf{\Omega}'$ in order of Hamming distance reduces the number of decodings and hashes that have to be performed.

Algorithm BioSCOM.Enroll = NaiveSCOM.Enroll

Algorithm BioSCOM.Verify

```
1. Read $P' = (\mathbf{\Omega}', G')$.
2. Measure the fresh biometric $X' \in \{0,1\}^n$.
3. Compute $F' = \mathtt{Syn}\, X'$.
4. For $j \in \{1,\dots,m\}$ do: $d_j = d_{\rm H}(F', \Omega'_j)$.
5. Make a permutation $\lambda$ that sorts $(d_j)_{j=1}^m$ in ascending order.
6. Let $\tilde{\mathbf{\Omega}} = \lambda(\mathbf{\Omega}')$.
7. Let $j = 0$.
8. Increase $j$. If $j = m+1$ then abort with failure.
9. Try to compute $E_j = \mathtt{SDec}(F' \oplus \tilde{\Omega}_j)$.
   If the decoding fails then goto 8.
10. $\hat{X}_j = X' \oplus E_j$.
11. If $G' \neq f(\mathbf{\Omega}' || \hat{X}_j)$ then goto 8.
12. Accept.
```

Remark: There are many alternative ways to organize the verification. For instance, in step 6 the vector $\mathbf{\Omega}'$ does not have to be physically permuted; permutation of the indices $\{1,\cdots,m\}$ is more efficient. Also, steps 4 and 5 can be combined to efficiently create an ordered list while computing the Hamming distances.

The computational workload of BioSCOM.Verify consists of: m Hamming distance computations; sorting of $\mathbf{\Omega}'$ (time $\propto m$ if steps 4 and 5 are combined, e.g. with an algorithm resembling Bucket Sort); a number of decodings and hashes that depends on the amount of noise in X'. Note that the timing side channel reveals only the amount of noise.

We propose that an LDPC code is used with column weight 3, i.e. three nonzero entries in each column of the parity check matrix. On the one hand, this allows for good error correction capabilities. On the other hand, one bit flip in X causes only 3 bit flips in $\mathtt{Syn}\, X$. Let us model the noise in X' as a binary symmetric channel with bit error rate (BER) β. Then, roughly speaking[3], $d_{\rm H}(\mathtt{Syn}\, X', \mathtt{Syn}\, X)$ is binomial-distributed with expectation $3n\beta$ and standard deviation $3\sqrt{n\beta(1-\beta)}$. A $W^{\rm fake}$ can be roughly modeled as a random string of length $n-k$. *Its* Hamming distance to $\mathtt{Syn}\, X$ is binomial-distributed with expectation $(n-k)/2$ and standard deviation $\frac{1}{2}\sqrt{n-k}$. The two distributions can be reliably distinguished if $3n\beta$ lies sufficiently below $(n-k)/2$, i.e. $\frac{k}{n}$ sufficiently below $1-6\beta$. Examples are given in Section 7.

5.3 Fuzzy Extractor

Below we present a pair of algorithms (FESCOM.Enroll, FESCOM.Reconstruct) for use as a Fuzzy Extractor in the POK scenario. This is a minor modification of (NaiveSCOM.Enroll, BioSCOM.Verify); the only difference is the derivation of a key Q from X and auxiliary public randomness A, with the use of the extractor function g.

[3]We ignore collisions. The given expressions are upper bounds.

Algorithm FESCOM.Enroll

```
1. Measure the POK output $X \in \{0,1\}^n$.
2. Generate random $A$. Compute $Q = g(X, A)$.
3. Compute helper data $W = \mathtt{Syn}\, X$.
4. For $j \in \{1,\dots,m-1\}$: Draw $W_j^{\rm fake} \in \{0,1\}^{n-k}$
   from the distribution of $\mathtt{Syn}\, X$.
5. Uniformly draw $Z \in \{1,\dots,m\}$.
6. Construct a vector
   $\mathbf{\Omega} = (W_1^{\rm fake},\cdots,W_{Z-1}^{\rm fake}, W, W_Z^{\rm fake},\cdots,W_{m-1}^{\rm fake})$.
7. Compute $G = f(\mathbf{\Omega}||A||X)$.
8. Store public data $P = (\mathbf{\Omega}, A, G)$.
```

Algorithm FESCOM.Reconstruct

```
1. Read $P' = (\mathbf{\Omega}', A', G')$.
2. Measure the POK output $X' \in \{0,1\}^n$.
3. Compute $F' = \mathtt{Syn}\, X'$.
4. For $j \in \{1,\dots,m\}$ do: $d_j = d_{\rm H}(F', \Omega'_j)$.
5. Make a permutation $\lambda$ that sorts $(d_j)_{j=1}^m$ in ascending order.
6. Let $\tilde{\mathbf{\Omega}} = \lambda(\mathbf{\Omega}')$.
7. Let $j = 0$.
8. Increase $j$. If $j = m+1$ then abort with failure.
9. Try to compute $E_j = \mathtt{SDec}(F' \oplus \tilde{\Omega}_j)$.
   If the decoding fails then goto 8.
10. $\hat{X}_j = X' \oplus E_j$.
11. If $G' \neq f(\mathbf{\Omega}'||A'||\hat{X}_j)$ then goto 8.
12. $\hat{Q} = g(\hat{X}_j, A')$.
```

6 Security analysis of the SCOM

We investigate how much information about X is revealed to the adversary by showing him $\mathbf{\Omega}$. In principle we should be looking at the leakage from the whole public data P, but there one hits a snag: information-theoretically there is no such thing as a one-way function. The hash G hides its input *in practice*, but information-theoretically speaking G reveals Z and W to the adversary. The leakage from $\mathbf{\Omega}$ is a better way to represent the adversary's actual workload than the leakage from $\mathbf{\Omega}$ and G. In effect, we will model the hash function as if it is perfectly hiding.

In the biometrics scenario, the relevant quantity to look at is Shannon entropy. (One might argue that min-entropy is more important, but since we do not have the stringent requirements that cryptographic keys have to satisfy[4], we will stick to Shannon entropy.) The relevant quantity in the POK scenario is the Rényi entropy H_2, which features in the ε-extractable randomness (3). We show results for both scenarios. In Section 7 we investigate memory requirements.

6.1 Leakage in terms of Shannon entropy

We first present a lemma that allows us to relate the leakage $I(X;\mathbf{\Omega})$ to the collisions in $\mathbf{\Omega}$.

[4]Most biometrics cannot be kept secret, since it is possible to measure them surreptitiously.

Lemma 6 *Consider the algorithm BioSCOM.Enroll or FESCOM.Enroll. Let $t(W, \boldsymbol{\Omega})$ denote the number of entries in $\boldsymbol{\Omega}$ equal to W, i.e. $t(W, \boldsymbol{\Omega}) = |\{j : \Omega_j = W\}|$. Then*

$$\mathsf{H}(X|\boldsymbol{\Omega}) = \mathsf{H}(X|W) + \log m - \mathbb{E}_{w\boldsymbol{\omega}} \log t(w, \boldsymbol{\omega}). \quad (9)$$

Proof: We start from Lemmas 3 and 4, which together give $\mathsf{H}(X|\boldsymbol{\Omega}) = \mathsf{H}(X|W) + \log m - \mathsf{H}(Z|W\boldsymbol{\Omega})$. Next we write $\mathsf{H}(Z|W\boldsymbol{\Omega}) = \mathbb{E}_{w\boldsymbol{\omega}}\mathsf{H}(Z|W = w, \boldsymbol{\Omega} = \boldsymbol{\omega})$. The uncertainty about Z given $W = w$ and $\boldsymbol{\Omega} = \boldsymbol{\omega}$ is caused by the fact that there can be multiple occurrences of the string $w \in \{0,1\}^{n-k}$ in $\boldsymbol{\omega}$; the number of occurrences is $t(w, \boldsymbol{\omega})$, and each of them is equally probable from the attacker's point of view. Hence $\mathsf{H}(Z|W = w, \boldsymbol{\Omega} = \boldsymbol{\omega}) = \log t(w, \boldsymbol{\omega})$. □

Theorem 1 *Consider the algorithm BioSCOM.Enroll or FESCOM.Enroll. The conditional entropy $\mathsf{H}(X|\boldsymbol{\Omega})$ can be bounded from below as*

$$\mathsf{H}(X|\boldsymbol{\Omega}) \geq \mathsf{H}(X|W) + \log m - \frac{m-1}{\ln 2}\mathbb{E}_w q_w. \quad (10)$$

Proof: We start from Lemma 6. We write $t(w, \boldsymbol{\omega}) = 1 + u(w, \boldsymbol{\omega})$ and use $\ln(1 + u) \leq u$. This gives $\mathbb{E}_{w\boldsymbol{\omega}} \log t(w, \boldsymbol{\omega}) \leq \frac{1}{\ln 2}\mathbb{E}_{w\boldsymbol{\omega}} u(w, \boldsymbol{\omega})$. For given w, the u is binomial-distributed with parameters $m-1$ and q_w. (See Section 2 for the notation q.) Thus we have $\mathbb{E}_{w\boldsymbol{\omega}} u(w, \boldsymbol{\omega}) = \mathbb{E}_w[(m - 1)q_w]$. □
The probability q_w is typically of the order $1/2^{n-k}$ if X is not too strangely distributed. Hence the last term in (10) is a small correction term if $m < 2^{n-k}$. Eq. (10) confirms the intuitive idea that the attacker's effort increases by a factor $\approx m/2$. Note that the bound in Theorem 1 is far from tight at large m. The following result is tighter at large m.

Theorem 2 *Consider the algorithm BioSCOM.Enroll or FESCOM.Enroll. The conditional entropy $\mathsf{H}(X|\boldsymbol{\Omega})$ can be bounded from below as*

$$\mathsf{H}(X|\boldsymbol{\Omega}) \geq \mathsf{H}(X) - \frac{1}{m} \cdot \frac{2^{n-k} - 1}{\ln 2}. \quad (11)$$

Proof: As in the proof of Theorem 1, we write $t = 1 + u$. Furthermore we split u into its expectation value (at fixed w) and a deviation: $u = (m - 1)q_w + \delta$, where $\mathbb{E}_\delta \delta = 0$. Then

$$\begin{aligned}\mathbb{E}_{w\boldsymbol{\omega}} \log t &= \mathbb{E}_w \log[mq_w] + \mathbb{E}_w \log[1 + \frac{1 - q_w}{mq_w}] \\ &\quad + \mathbb{E}_{w\delta} \log(1 + \frac{\delta}{1 + [m - 1]q_w}). \quad (12)\end{aligned}$$

Next we use $\ln(1 + z) \leq z$ twice, and $\mathbb{E}_\delta \delta = 0$, to get

$$\begin{aligned}\mathbb{E}_{w\boldsymbol{\omega}} \log t &\leq \log m + \mathbb{E}_w \log q_w + \frac{1}{\ln 2}\mathbb{E}_w \frac{1 - q_w}{mq_w} \\ &= \log m - \mathsf{H}(W) + \frac{1}{m \ln 2}(-1 + \sum_z \frac{q_z}{q_z}) \\ &= \log m - \mathsf{H}(W) + \frac{2^{n-k} - 1}{m \ln 2}. \quad (13)\end{aligned}$$

We substitute (13) into Lemma 6 and use $\mathsf{H}(X|W) + \mathsf{H}(W) = \mathsf{H}(X)$. □
In the limit $m > 2^{n-k}$, the $\frac{1}{m}$ term in Theorem 2 is a small correction term; we see that $\boldsymbol{\Omega}$ then hardly leaks anything about X, as we would expect intuitively, since $\boldsymbol{\Omega}$ is likely to contain almost all possible strings of length $n - k$.

6.2 Leakage in terms of Rényi entropy

We present a bound on $\mathsf{H}_2(X|\boldsymbol{\Omega})$ that is useful for large m. We observe that for the adversary each entry in $\boldsymbol{\omega}$ is equally likely to be the correct one. Thus, his knowledge about x can be parametrized as a probability distribution that is conditioned on each of the entries ω_j with equal probability $1/m$. (Remember that the fake helper data are drawn from the same distribution as $\texttt{Syn}\, X$). This gives

$$p_{x|\boldsymbol{\omega}} = \frac{1}{m}\sum_{j=1}^{m} p_{x|\omega_j}. \quad (14)$$

Based on (14) we obtain the following result.

Theorem 3 *Consider the algorithm BioSCOM.Enroll or FESCOM.Enroll. The conditional Rényi entropy $\mathsf{H}_2(X|\boldsymbol{\Omega})$ can be bounded from below as*

$$\mathsf{H}_2(X|\boldsymbol{\Omega}) \geq \mathsf{H}_2(X) - \frac{1}{m \ln 2}\left[\frac{\mathbb{E}_w \sum_x p^2_{x|w}}{\sum_x p^2_x} - 1\right]. \quad (15)$$

Proof:

$$\begin{aligned}\mathsf{H}_2(X|\boldsymbol{\Omega}) &= -2 \log \mathbb{E}_{\boldsymbol{\omega}} \sqrt{\textstyle\sum_x p^2_{x|\boldsymbol{\omega}}} \quad (16) \\ &\geq -2 \log \sqrt{\mathbb{E}_{\boldsymbol{\omega}} \textstyle\sum_x p^2_{x|\boldsymbol{\omega}}} \quad (17)\end{aligned}$$

$$= -\log \mathbb{E}_{\boldsymbol{\omega}} \sum_x \frac{1}{m^2} \sum_{a,b=1}^{\ell} p_{x|\omega_a} p_{x|\omega_b} \quad (18)$$

$$= -\log \frac{1}{m^2}\sum_x \left[\sum_a \mathbb{E}_{\boldsymbol{\omega}} p^2_{x|\omega_a} + \sum_{a,b:a\neq b} \mathbb{E}_{\boldsymbol{\omega}} p_{x|\omega_a} p_{x|\omega_b}\right] \quad (19)$$

$$= -\log \sum_x \left[\frac{1}{m}\mathbb{E}_w p^2_{x|w} + [1 - \frac{1}{m}]p^2_x\right] \quad (20)$$

$$= -\log \left[(\sum_x p^2_x)(1 + \frac{\mathbb{E}_w \sum_x p^2_{x|w} - \sum_x p^2_x}{m \sum_x p^2_x})\right] \quad (21)$$

$$= \mathsf{H}_2(X) - \log(1 + \frac{\mathbb{E}_w \sum_x p^2_{x|w} - \sum_x p^2_x}{m \sum_x p^2_x}). \quad (22)$$

In (17) we used Jensen's inequality. In (18) we substituted (14). Finally (15) is obtained from (22) by using $\log(1+z) = \ln(1 + z)/\ln 2 \leq z/\ln 2$. □
Remark: If X is not too far from uniform, then the $\frac{1}{m}$-term in Theorem 3 is of order $2^{n-k}/m$, i.e. the same order of magnitude as the $\frac{1}{m}$-term in Theorem 2.
When spammed helper data $\boldsymbol{\Omega}$ is used instead of W, the entropy $\mathsf{H}_2(X|W)$ in the extractable randomness formula (3) can be replaced by the (much) larger number $\mathsf{H}_2(X|\boldsymbol{\Omega})$.

7 Numerical examples

In the biometrics scenario, a large amount of storage space is available per enrolled person, since the public data P is usually stored in a dedicated database. Blowing up the database by a large factor could be feasible. Furthermore, the original W is a short string to start with.
In the POK scenario, the public data is usually stored on the device that contains the POK. This device has to be cheap; hence nonvolatile memory may become an issue.

In the examples below we consider a Binary Symmetric Channel (BSC) with bit error rate (BER) β.

Example 1. Consider a uniform biometric X with $n = 127$ and $\beta = 2.2\%$. (Such a low BER can be obtained by preprocessing, e.g. reliable component selection). We look at the NaiveSCOM scheme employing a BCH code $(n, k, t) = (127, 50, 13)$, i.e. the code can correct 13 errors, which at the given BER corresponds to a decoding error probability of less than 10^{-6}. For $m = 1$ the security is $\mathsf{H}(X|\mathbf{\Omega}) = k = 50$ bits. In order to gain δ bits of security we need $m \approx 2^{\delta+1}$, which requires a storage space of $2^{\delta+1}(n-k) = 2^{\delta-5.7}$KB. One MB available per enrolled user would allow for $\delta = 15.7$. Efficient syndrome decoders exist for BCH codes [11].

Example 2: *POK storing a symmetric key.* Consider a non-uniform X with $n = 816$ and $\beta = 4.0\%$, whose distribution is such that $\mathtt{Syn}\, X$ is approximately uniform. We look at the FESCOM scheme employing an LDPC code with the following parameters [21]: column weight 3; rate 1/2 ($k = 408$). The decoding error probability[5] is around 10^{-5}. In order to achieve δ extra bits of security, the required storage is $2^{\delta+1}(n-k)$ bits $\approx 2^{\delta-3.3}$KB.

The column weight of the parity check matrix is 3, which means that each bit flip in X' causes 3 bit flips in $\mathtt{Syn}\, X'$. We have $d_{\rm H}(\mathtt{Syn}\, X', \mathtt{Syn}\, X) \leq 3 \cdot d_{\rm H}(X', X)$. The expected number of bit flips in X' is 32.6, with a standard deviation of 5.6, which translates to less than 97.8 and 16.8 respectively in $\mathtt{Syn}\, X'$.

The Hamming distance between $\mathtt{Syn}\, X$ (or $\mathtt{Syn}\, X'$) and a randomly generated 408-bit string, on the other hand, is on average 204 bits with a standard deviation of 10.1.

We may consider using a threshold of ≈ 161 bit flips in the syndrome to distinguish between the real helper data and the fakes. The probability $\Pr[d_{\rm H}(\mathtt{Syn}\, X', \mathtt{Syn}\, X) > 161]$ is less than $1.6 \cdot 10^{-4}$, while The probability that a random 408-bit string is removed from $\mathtt{Syn}\, X$ (or $\mathtt{Syn}\, X'$) by fewer than 161 bit flips is $1 \cdot 10^{-5}$. Hence, even if $\mathbf{\Omega}$ has of order 10^5 entries, the correct helper data clearly stands out to the legitimate user.

Example 3: *POK storing an RSA prime.* Similar to Example 2. $n = 8000$; $k = 4000$; $\beta = 5.4\%$; storage $2^{\delta-0.03}$KB. The number of bit flips in $\mathtt{Syn}\, X'$ has average ≤ 1296 and standard deviation ≤ 60.6, whereas a random 4000-bit string has average Hamming distance 2000 with standard deviation 31.6. We have $\Pr[d_{\rm H}(\text{random string}, \mathtt{Syn}\, X) \leq 1850] = 1 \cdot 10^{-6}$ and $\Pr[d_{\rm H}(\mathtt{Syn}\, X', \mathtt{Syn}\, X) > 1590] \leq 1 \cdot 10^{-6}$, again a clear separation between the real entry and the fake ones.

8 Generative SCOM

The list $\mathbf{\Omega}$ can grow very large if one wishes to reduce a significant part of the leakage $I(W; X)$. Below we present a method for generating $\mathbf{\Omega}$ on the fly. This changes the security$\leftrightarrow$storage tradeoff into a security$\leftrightarrow$processing tradeoff. Given the existence of Pseudo-Random Number Generators (PRNGs) that output more than one byte per clock cycle [1, 2], it can be faster to *generate* $\mathbf{\Omega}$ than to fetch it from nonvolatile memory.

[5]In the literature on LDPC codes, the performance is often given for the Additive White Gaussian Noise channel. For this channel, soft information is available, and up to 6% BER can be tolerated with the given decoding failure probability of 10^{-5}. We translate this (somewhat loosely) to $\beta = 4.0\%$ on the BSC.

8.1 Construction

The construction relies on a fast PRNG γ that generates $(n-k)$-bit strings, given a seed S. The i'th string derived from S is denoted as $\gamma^i(S) \in \{0,1\}^{n-k}$. The verification algorithm uses thresholds $\theta_0, \ldots, \theta_N$ on the Hamming distance to distinguish between likely candidates (small $d_{\rm H}$) and unlikely candidates (large $d_{\rm H}$). Unlikely candidates are postponed. The thresholds satisfy $\theta_0 = 0$ and $\theta_0 < \theta_1 < \cdots < \theta_N$. The length m is considered to be a fixed system parameter.

```
Algorithm GenSCOM.Enroll
1. Measure $X \in \{0,1\}^n$.
2. Compute $W = \mathtt{Syn}\, X$.
3. Uniformly draw $Z \in \{1, \ldots, m\}$.
4. Uniformly draw a seed $S$.
5. Compute the mask $B = W \oplus \gamma^Z(S)$.
6. Compute $G = f(S||B||X)$.
7. Store public data $P = (S, B, G)$.
```

```
Algorithm GenSCOM.Verify
1. Set Auth=False.
2. Read $P' = (S', B', G')$.
3. Measure $X' \in \{0,1\}^n$.
4. Compute $F = B' \oplus \mathtt{Syn}\, X'$.
5. For $r = 1$ to $N$
6. For $i = 1$ to $m$
    (a) $\Delta_i = F \oplus \gamma^i(S')$.
    (b) If HammWeight$(\Delta_i) \notin \{\theta_{r-1}, \ldots, \theta_r - 1\}$
        then Next $i$.
    (c) Try to compute $E_i = \mathtt{SDec}\, \Delta_i$.
        If the decoding fails then Next $i$.
    (d) $\hat{X}_i = X' \oplus E_i$.
    (e) If $f(m||S'||B'||\hat{X}_i) = G'$ then set Auth=True.
7. Next $i$.
8. If Auth=False then Next $r$.
9. Return Auth.
```

- In GenSCOM.Verify, the mask B' is added to $\mathtt{Syn}\, X'$ in step 4 for efficiency reasons. (The alternative would be to add the mask inside the i-loop.)
- The full i-loop is completed even after the correct entry is found. This is done in order to thwart timing side channel attacks. The timing reveals only the number of completed r-rounds, and the total number of decodings and hashings of fake entries; i.e. the amount of noise is revealed.
- We have shown only the scheme for the biometrics scenario. It can be trivially adapted to the POK scenario.
- The distribution of $d_{\rm H}(W^{\rm fake}, \mathtt{Syn}\, X')$ is practically the same as the distribution of $d_{\rm H}(W^{\rm fake}, \mathtt{Syn}\, X)$ as discussed in Section 5.2. Hence the same analysis applies.

8.2 Security analysis of GenSCOM

In the GenSCOM scheme, the role of the list $\mathbf{\Omega}$ is taken by the generated sequence $\mathbf{\Omega}^{\rm gen} = (B \oplus \gamma^i(S))_{i=1}^m$, with $\mathbf{\Omega}_Z^{\rm gen} =$

$\mathtt{Syn}\, X$. The quantity of interest for the security is $\mathsf{H}(X|SB)$, since S and B are stored as public data.

Lemma 7 *Consider GenSCOM.enroll. It holds that*

$$\mathsf{H}(X|SB) = \mathsf{H}(X|\boldsymbol{\Omega}^{\rm gen}). \tag{23}$$

<u>*Proof:*</u> We have $\mathsf{H}(X|BS) = \mathsf{H}(X|\boldsymbol{\Omega}^{\rm gen}S)$ since $\boldsymbol{\Omega}^{\rm gen}$,$S$ together contain the same information as B, S. Finally we note that if $\boldsymbol{\Omega}^{\rm gen}$ is already known, then S reveals no extra information about X. □

Theorem 4 *Consider GenSCOM.Enroll. Let $U \in \{0,1\}^{n-k}$ be a uniform random variable. Then*

$$\begin{aligned}\mathsf{H}(X|SB) \quad &\geq \quad \mathsf{H}(X|W) + \log m - \frac{m-1}{2^{n-k}\ln 2} \\ &-(1-\frac{1}{m})\Big[D(W||U) + D(U||W)\Big]. \end{aligned} \tag{24}$$

<u>*Proof:*</u> We start from Lemma 7 and use Lemma 4, replacing $\boldsymbol{\Omega} \to \boldsymbol{\Omega}^{\rm gen}$. We have $\mathsf{H}(Z) = \log m$. For the collision penalty we get $\mathsf{H}(Z|W\boldsymbol{\Omega}^{\rm gen}) = \mathbb{E}_{w\boldsymbol{\omega}} \log(1+u) \leq \mathbb{E}_{w\boldsymbol{\omega}} u/\ln 2$ just as in the proof of Theorem 1. Now, however, the entries are uniformly generated instead of having distribution q_w. This gives $\mathbb{E}_{w\boldsymbol{\omega}} u = (m-1)/2^{n-k}$. Finally, for the distribution mismatch penalty $I(Z;\boldsymbol{\Omega}^{\rm gen})$ we use Lemma 5 with $W^{\rm fake} = U$ and uniform Z. □
Note: the expression $D(W||U) + D(U||W)$ can be written as $\sum_w (2^{k-n} - q_w) \log \frac{1}{q_w}$.

8.3 GenSCOM alternatives

If we are unlucky, GenSCOM.Verify makes multiple passes through the whole list $\boldsymbol{\Omega}$. As an alternative algorithm, one could temporarily store unlikely candidates before trying them. A greedy algorithm would work e.g. as follows:

1. Store candidates whose Hamming weight is lower than a threshold, in ascending order, until the buffer is full.

2. For the most promising candidate, do the decoding and, if applicable, the hashing step. If the candidate fails then go back to 1.

It is possible to reduce the running time while still retaining resistance against the timing side channel: The order in which the $\boldsymbol{\Omega}$ entries are generated by the PRNG may be randomized to some extent. Then it is no longer necessary to complete the full i-loop. This approach requires storing multiple seeds. Note that all the SCOM algorithms are easy to parallelize.

9 Discussion / related work

We have proposed the Spammed Code Offset Method, in which the adversary gets spammed with bogus helper data. For small spam factor m, the security is increased by roughly $\log m$ bits. While the workload of the adversary is increased by a factor $m/2$, the workload of the legitimate party stays manageable: only few decodings and hashings are needed. This is achieved by using Hamming distance in syndrome space as a fast candidate selection criterion, where the use of an LDPC code makes sure that a small distance between X and X' translates to a small distance in syndrome space; in case of an LDPC code with column weight 3, there are no more than 3 bit flips in $\mathtt{Syn}\, X$ per bit flip in X.

The SCOM works best if the fake helper data has the same distribution as the real one. Compared to other helper data schemes, this requires more precise knowledge of the source X. If the distributions are not the same, then the entropy of W suffers a distribution mismatch penalty $I(Z;\boldsymbol{\Omega})$ (Lemma 4).

The SCOM provides a new kind of trade-off: a more effective use of source entropy is achieved at the price of storage or computation effort at reconstruction. This is especially interesting in applications where the source entropy is limited.

In the POK scenario it depends on various system parameters whether it makes sense to use the SCOM. If the available storage/CPU resources in the device are limited and there is ample entropy in X, then the ordinary COM suffices.

In the biometrics case it is especially important to eliminate the leakage $I(X;W)$, since the entropy of X is usually rather low and has to be maximally exploited. Fortunately it is easier to meet the memory requirements in this scenario.

The timing side channel does not leak information about the location Z. NaiveSCOM.Reconstruct parses the whole list; In BioSCOM, FESCOM and GenSCOM the timing reveals only the amount of noise in X'.

The idea of adding fake entries is not new in the context of noisy data, but to the best of our knowledge we are the first to apply it in helper data space. In [5] fake fingerprint minutiae are added to the stored template. A comparison *in plaintext* decides if a fresh fingerprint matches a sufficient number of stored minutiae. The 'Fuzzy Vault' scheme [18] adds a large amount of 'chaff' points to (e.g. biometric) data points contained in X. Our scheme differs significantly in two ways: (i) The amount of chaff in the Fuzzy Vault is necessarily large and does not allow for a security vs. storage trade-off at small amounts of storage; (ii) The Fuzzy Vault simultaneously hides the measurement X and a secret key.

The use of LDPC codes in the context of biometrics is not new. They have good error-correcting properties and easy to implement decoders. Furthermore, the belief propagation in the decoder lends itself to handling input bits of unequal reliability [33]. LDPC codes have also been proposed as a noise-tolerant hash function [3]. To the best of our knowledge, the SCOM is the first scheme that uses an LDPC code to filter out decoy entries.

As future work we mention experiments with various LDPC codes. Another interesting issue to look at is the cross-linkability between biometric templates in different databases. We remark that the process of recognizing which part of the data is 'real' is closely related to biometric *identification*: the fakes can be thought of as the biometrics of other people. When m is large, it becomes much harder for an adversary to decide if templates in different databases belong to the same person, since the decoys are likely to cause false matches.

Acknowledgements
We thank Ruud Pellikaan, Ludo Tolhuizen, Joep de Groot and Jean-Paul Linnartz for useful discussions. We thank the anonymous reviewers for their comments. Part of this work was funded by the European project INFSO-ICT-284833 (PUFFIN).

Appendix: Proof of Lemma 5

We introduce shorthand notation $\varphi_w = \Pr[W^{\text{fake}} = w]$.

$$
\begin{aligned}
I(Z;\mathbf{\Omega}) &= -\mathbb{E}_{z\boldsymbol{\omega}} \log \frac{\Pr[\mathbf{\Omega}=\boldsymbol{\omega}]\Pr[Z=z]}{\Pr[\mathbf{\Omega}=\boldsymbol{\omega}, Z=z]} \\
&= -\mathbb{E}_{z\boldsymbol{\omega}} \log \frac{\Pr[\mathbf{\Omega}=\boldsymbol{\omega}]}{\Pr[\mathbf{\Omega}=\boldsymbol{\omega}|Z=z]} \\
&= -\mathbb{E}_{z\boldsymbol{\omega}} \log \mathbb{E}_a \frac{\Pr[\mathbf{\Omega}=\boldsymbol{\omega}|Z=a]}{\Pr[\mathbf{\Omega}=\boldsymbol{\omega}|Z=z]} \\
&= -\mathbb{E}_{z\boldsymbol{\omega}} \log \mathbb{E}_a \frac{q_{\omega_a}\varphi_{\omega_z}}{q_{\omega_z}\varphi_{\omega_a}} && (25) \\
&\le -\mathbb{E}_{z\boldsymbol{\omega}} \mathbb{E}_a \log \frac{q_{\omega_a}\varphi_{\omega_z}}{q_{\omega_z}\varphi_{\omega_a}} && (26) \\
&= -\mathbb{E}_z \sum_{a\neq z}\pi_a \sum_{\omega_z} q_{\omega_z} \sum_{\omega_a}\varphi_{\omega_a} \log \frac{q_{\omega_a}\varphi_{\omega_z}}{q_{\omega_z}\varphi_{\omega_a}} && (27) \\
&= -\mathbb{E}_z \sum_{a\neq z}\pi_a \Big[\sum_{\omega_z} q_{\omega_z} \log \frac{\varphi_{\omega_z}}{q_{\omega_z}} + \sum_{\omega_a}\varphi_{\omega_a} \log \frac{q_{\omega_a}}{\varphi_{\omega_a}}\Big] \\
&= \mathbb{E}_z \sum_{a\neq z}\pi_a [D(W||W^{\text{fake}}) + D(W^{\text{fake}}||W)] && (28) \\
&= \mathbb{E}_z (1-\pi_z)[D(W||W^{\text{fake}}) + D(W^{\text{fake}}||W)]. && (29)
\end{aligned}
$$

In (25) we used the fact that in $\Pr[\mathbf{\Omega}=\boldsymbol{\omega}|Z=z]$ and $\Pr[\mathbf{\Omega}=\boldsymbol{\omega}|Z=a]$ all the components $\ell \neq a, z$ have $\Pr[\Omega_\ell = \omega] = \Pr[W^{\text{fake}} = \omega] = \varphi_\omega$, which leads to cancellation in the fraction. In (26) we used Jensen's inequality to write $\log \mathbb{E}_a \le \mathbb{E}_a \log$. In (27) we wrote out $\mathbb{E}_{\boldsymbol{\omega}}$, using the fact that only the components ω_z and ω_a are relevant, and that ω_z is the actual helper data W while ω_a is fake. In (28) we used the definition of the KL distance.

References

[1] `http://gjrand.sourceforge.net/`.

[2] `http://www.digicortex.net/node/22`.

[3] M. Baldi, M. Bianchi, F. Chiaraluce, J. Rosenthal, and D. Schipani. On fuzzy syndrome hashing with LDPC coding. In *Int. Symposium on Applied Sciences in Biomedical and Communication Technologies*, pages 24:1–24:5. ACM, 2011.

[4] B. Barak, Y. Dodis, H. Krawczyk, O. Pereira, K. Pietrzak, F.-X. Standaert, and Y. Yu. Leftover Hash Lemma, revisited. In *CRYPTO*, volume 6841 of *LNCS*, pages 1–20. Springer, 2011.

[5] C. Barral. *Biometrics & Security: Combining Fingerprints, Smart Cards and Cryptography.* PhD thesis, École Polytechnique Fédérale de Lausanne', 2010.

[6] C. Böhm and M. Hofer. *Physical Unclonable Functions in Theory and Practice.* Springer, 2013.

[7] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith. Secure remote authentication using biometric data. In *Eurocrypt 2005*, volume 3494 of *LNCS*, pages 147–163. Springer-Verlag, 2005.

[8] J.L. Carter and M.N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[9] R. Cramer, Y. Dodis, S. Fehr, C. Padró, and D. Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 471–488, 2008.

[10] J.A. de Groot, B. Škorić, N. de Vreede, and J.-P. Linnartz. Information leakage of continuous-source Zero Secrecy Leakage Helper Data Schemes. `http://eprint.iacr.org/2012/566`, 2012.

[11] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy Extractors: how to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[12] Y. Dodis, M. Reyzin, and A. Smith. Fuzzy Extractors: How to generate strong keys from biometrics and other noisy data. In *Eurocrypt 2004*, volume 3027 of *LNCS*, pages 523–540. Springer-Verlag, 2004.

[13] S. Fehr and S. Berens. The conditional Rényi entropy, 2013.

[14] M. Franceschini, G. Ferrari, and R. Raheli. *LDPC Coded Modulations.* Springer, 2009.

[15] R.G. Gallager. Low-Density Parity Check Codes. Monograph. MIT., 1963.

[16] B. Gassend. Physical Random Functions. Master's thesis, Massachusetts Institute of Technology, 2003.

[17] J. Håstad, R. Impagliazzo, L.A. Levin, and M. Luby. Construction of pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[18] A. Juels and M. Sudan. A fuzzy vault scheme. In *IEEE International Symposium on Information Theory (ISIT) 2002*, page 408. IEEE Press.

[19] A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *ACM Conference on Computer and Communications Security (CCS) 1999*, pages 28–36, 1999.

[20] J.-P. Kaps, K. Yüksel, and B. Sunar. Energy scalable universal hashing. *IEEE Trans. Computers*, 54(12):1484–1495, 2005.

[21] D. MacKay. Gallager code resources. `http://wol.ra.phy.cam.ac.uk/mackay/CodesFiles.html`.

[22] R. Maes. *Physically Unclonable Functions: Constructions, Properties and Applications.* Springer, 2013.

[23] R. Renner and S. Wolf. Smooth Rényi entropy and applications. In *IEEE International Symposium on Information Theory (ISIT) 2004*, page 233, 2004.

[24] R. Renner and S. Wolf. Simple and tight bounds for information reconciliation and privacy amplification. In *Asiacrypt 2005*, volume 3788 of *LNCS*, pages 199–216. Springer-Verlag, 2005.

[25] T. Richardson and R. Urbanke. *Modern Coding Theory.* Cambridge University Press, 2008.

[26] A.-R. Sadeghi and D. Naccache, editors. *Towards hardware-intrinsic security.* Springer, 2010.

[27] D.R. Stinson. Universal hashing and authentication codes. *Designs, Codes, and Cryptography*, 4:369–380, 1994.

[28] P. Tuyls, G.-J. Schrijen, B. Škorić, J. van Geloven, R. Verhaegh, and R. Wolters. Read-proof hardware from protective coatings. In *Cryptographic Hardware and Embedded Systems (CHES) 2006*, volume 4249 of *LNCS*, pages 369–383. Springer-Verlag, 2006.

[29] P. Tuyls, B. Škorić, and T. Kevenaar. *Security with Noisy Data: Private Biometrics, Secure Key Storage and Anti-Counterfeiting.* Springer, London, 2007.

[30] A. van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann. Reverse Fuzzy Extractors: enabling lightweight mutual authentication for PUF-enabled RFIDs. In *Financial Cryptography and Data Security*, volume 7397 of *LNCS*, pages 374–389, 2012.

[31] E.A. Verbitskiy, P. Tuyls, C. Obi, B. Schoenmakers, and B. Škorić. Key extraction from general nondiscrete signals. *IEEE Transactions on Information Forensics and Security*, 5(2):269–279, 2010.

[32] B. Škorić, C. Obi, E.A. Verbitskiy, and B. Schoenmakers. Sharp lower bounds on the extractable randomness from non-uniform sources. *Information and Computation*, 209:1184–1196, 2011.

[33] Y. Wang, S. Rane, and A. Vetro. Leveraging reliable bits: ECC design considerations for practical secure biometric systems. In *Workshop on Information Forensics and Security*, pages 71–75. IEEE, 2009.