The spammed code offset method

Citation for published version (APA):

Document status and date:
Published: 01/01/2013

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.
Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 18. Mar. 2024
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 Uncloneable 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 Uncloneable 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.](image1)

![Figure 2: The Code Offset Method employed as a Secure Sketch.](image2)

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 $\rho$. 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 $\text{Enc}$ and $\text{Dec}$ respectively. The COM takes $R$ uniformly drawn from $\{0,1\}^k$ and

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

(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
In Section 1.3 we introduce notation and assumptions. In Section 2 we introduce notation and assumptions. In Section 3 we discuss a version of the COM that does not need to keep the legitimate party's workload low. 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 GSOM (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 $d_H(x,y)$ stands for the Hamming distance between $x$ and $y$. The logarithm ‘log’ has base 2. The natural logarithm is ln. The unit vector $e_j$ 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 $C$ that has $n$-bit code words and $k$-bit messages. The encoding and decoding algorithms associated with this code are denoted as $Enc: \{0,1\}^k \rightarrow \{0,1\}^n$ and $Dec: \{0,1\}^n \rightarrow \{0,1\}^k$ respectively. The algorithm for computing the syndrome is denoted as $Syn: \{0,1\}^n \rightarrow \{0,1\}^{n-k}$. If the decoder is a syndrome decoder, then the mapping from a syndrome to a minimal error pattern is called $SDec: \{0,1\}^{n-k} \rightarrow \{0,1\}^n$.

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 $Q \in \{0,1\}^2$. We will use shorthand notation $p_{x=w} = Pr[X=x, W=w]$, $p_w = Pr[W=w]$ and $p_{x|w} = Pr[X=x|W=w]$, when it does not cause ambiguity. We define $q_z = Pr[Syn(X) = z]$. 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 $h$. 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 $H(X)$. Conditional Shannon entropy is denoted as $H(X|Y)$. 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. Biometric database for authentication.

   The adversary can read but not manipulate the public data $P$. His aim is to learn as much about $X$ as he can.\footnote{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.}

2. Secure key storage with a POK.

   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...
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 \(\text{SDec}\). 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 \(W = \text{Syn} X\).
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 \(\sigma = W' \oplus \text{Syn} X'\).
4. \(E = \text{SDec} \sigma\).
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: \(\sigma = W' \oplus \text{Syn} X' = \text{Syn} X \oplus \text{Syn} X' = \text{Syn} (X' \oplus X)\). 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 \(P'\), \(W'\), \(\zeta'\), \(h'\) 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 \(\text{Syn} L_j = e_j\).
   - Publicly store \(L = (L_j)_{j=1}^{n-k}\).

**Algorithm 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. \(E = V \oplus \text{Enc}(\text{Dec}(V))\).
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 \(\sum_{j \in L_j}\) 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 \(\sum_{j \in L_j} W_j L_j\) 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 \(\text{Dec}\) can be any decoder algorithm.
- An alternative (and security-wise equivalent) enrollment procedure would be to store \(\sum_{j \in L_j} \text{Syn}(X)\) instead of \(\text{Syn} X\) as helper data. That would constitute a deterministic version of the helper data \(X \oplus \text{Enc} R\) 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 $\Pr[\text{Syn} X = y]$ as $q_y$. Thus we have $\Pr[W = w] = q_w$.

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

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

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

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

$$I(W; X) = H(\text{Syn} X), \quad H(W; X) = 0 \quad H(X|W) = H(X) - H(\text{Syn} X).$$

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

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 $\ell$): $Q = g(X, A) \in \{0, 1\}^{\ell}$. 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 $\ell$ and the uniformity of $Q$ is given by the Leftover Hash Lemma (LHL) [17] and can be formulated as

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

with $L_\varepsilon(X, W) = H_2(X|W) + 1 - 2 \log \frac{1}{\varepsilon}$. \quad (3)

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

$$H_2(X|W) = -2 \log T_2(X|W) \quad T_2(X|W) = \mathbb{E}_w \sqrt{\sum_x p_x^2|w} = \sqrt{\sum_x p_x^2|w}, \quad (4)$$

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 $\varepsilon$, 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 $W_1^{\text{fake}}, \ldots, W_{m-1}^{\text{fake}}$ 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 $\Omega$ be a vector constructed from the real and the fake helper data as $\Omega = (W_1^{\text{fake}}, \ldots, W_{Z-1}^{\text{fake}}, W, W_{Z+1}^{\text{fake}}, \ldots, W_{m-1}^{\text{fake}})$. Then

$$H(X|\Omega) = H(X|W) + H(W|\Omega) - H(W, X|\Omega). \quad (5)$$

If $W$ is a function of $X$ only, then

$$H(X|\Omega) = H(X|W) + H(W|\Omega). \quad (6)$$

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

**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 $H(X|W)$, which is the ‘old’ result for an ordinary helper data scheme, and the contribution $H(W|\Omega)$ that arises from hiding $W$ in a random position in the list $\Omega$. The ‘hiding’ term $H(W|\Omega)$ can be further broken down as follows.

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

$$H(W|\Omega) = H(Z) - H(Z|W|\Omega) - I(Z; \Omega). \quad (7)$$

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

$$H(W|\Omega) = \log m - H(Z|W|\Omega). \quad (8)$$

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

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

The ‘distribution mismatch penalty’ occurs when the fake entries in \(\Omega\) do not look statistically the same as \(W\); then some information about \(Z\) can be obtained already from inspecting \(\Omega\).

At first sight (6) might seem to contradict the well known results from a 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 \(\Omega\) be defined as in Lemma 3. Let \(W_{\text{fake}}\) denote an arbitrary fake entry in \(\Omega\). Let \(\pi_z = \Pr[Z = z]\). Then

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

**Proof:** See Appendix. \(\square\)

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 SCOMs 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 \(W = \text{Syn} X\).
3. For \(j \in \{1, \ldots, m-1\}\): draw \(W_j = \{0,1\}^{n-k}\) from the distribution of \(\text{Syn} X\).
4. Uniformly draw \(Z \in \{1, \ldots, m\}\).
5. Construct a vector \(\Omega = (W_1, \ldots, W_m, W_{\text{fake}}, \ldots, W_{m-1})\).
6. Compute \(G = f(\Omega||X)\).
7. Store public data \(P = (\Omega, G)\).

---

\(^2\)It is possible to avoid collisions during the construction of \(\Omega\). We will not consider this approach since it has very limited benefit.

---

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 \(\Omega\). 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 \(X_j^\text{fake}\) from the distribution of \(X\) and then compute \(W_j = \text{Syn} X_j^\text{fake}\).

At reconstruction the public data may have been altered, which is why we use the notation \(P', \Omega', G'\) in step 1 of the reconstruction. A list \(L_1\) is made of \(\Omega\) entries that lead to successful decoding. The whole set \(L_1\) 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 \(L_2\) of entries whose \(X_j\) generates the correct hash. If \(P' = P\) and \(X' \approx X\) then typically there is only one candidate left in \(L_2\). If \(P' \neq P\) or \(X'\) is too noisy to be error-corrected, then typically \(L_2 = \emptyset\).

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 \(|L_1|\) hashes are computed in step 6. The choice of \(n, k, m\) may give rise to a long list \(L_1\). 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 \(L_1\), and possibly of \(L_2\). We do not consider this to be a security leak, since the \(|L_1|\) and \(|L_2|\) 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 \(\text{Syn} X'\) to \(\text{Syn} X\) and the other syndromes allows the device to heuristically re-order \(\Omega\) 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 \(\text{Syn} X\) and \(\text{Syn} X'\).
Processing $\Omega'$ 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 $P' = (\Omega', G')$.
2. Measure the fresh biometric $X' \in \{0, 1\}^n$.
3. Compute $F' = \text{Syn} X'$.
4. For $j \in \{1, \ldots, m\}$ do: $d_j = d_{\text{hit}}(F', \Omega_j')$.
5. Make a permutation $\lambda$ that sorts $(d_j)_{j=1}^m$ in ascending order.
6. Let $\tilde{\Omega} = \lambda(\Omega')$.
7. Let $j = 0$.
8. Increase $j$. If $j = m + 1$ then abort with failure.
9. Try to compute $E_j = \text{SDec}(F' \oplus \tilde{\Omega}_j)$. If the decoding fails then goto 8.
10. $\tilde{X}_j = X' \oplus E_j$.
11. If $G' \neq f(\Omega'||\tilde{X}_j)$ then goto 8.

**Remark:** There are many alternative ways to organize the verification. For instance, in step 6 the vector $\Omega'$ does not have to be physically permuted; permutation of the indices $\{1, \ldots, m\}$ 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 $\Omega'$ (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 depend 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 $\text{Syn} X$. Let us model the noise in $X'$ as a binary symmetric channel with bit error rate (BER) $\beta$. Then, roughly speaking, $d_{\text{hit}}(\text{Syn} X', \text{Syn} X)$ is binomial-distributed with expectation $3n\beta$ and standard deviation $\sqrt{3n(1-\beta)}$.

A $W^\text{fake}$ can be roughly modeled as a random string of length $n-k$. Its Hamming distance to $\text{Syn} X$ 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. $\frac{k}{n} \leq 6/3$. 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 $g$.

#### 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 $W = \text{Syn} X$.
4. For $j \in \{1, \ldots, m-1\}$: Draw $W^\text{fake}_j \in \{0, 1\}^{n-k}$ from the distribution of $\text{Syn} X$.
5. Uniformly draw $Z \in \{1, \ldots, m\}$.
6. Construct a vector $\Omega = (W^\text{fake}_1, \ldots, W^\text{fake}_{Z-1}, W^\text{fake}_Z, \ldots, W^\text{fake}_{m-1})$.
7. Compute $G = f(\Omega||A||X)$.
8. Store public data $P = (\Omega, A, G)$.

#### Algorithm FESCOM.Reconstruct

8. Increase $j$. If $j = m + 1$ then abort with failure.
9. Try to compute $E_j = \text{SDec}(F' \oplus \tilde{\Omega}_j)$. If the decoding fails then goto 8.
10. $\tilde{X}_j = X' \oplus E_j$.
11. If $G' \neq f(\Omega'||\tilde{X}_j)$ then goto 8.
12. $\hat{Q} = g(\tilde{X}_j, A')$.

### 6 Security analysis of the SCOM

We investigate how much information about $X$ is revealed to the adversary by showing him $\Omega$. In principle we should be looking at the leakage from the 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 $\Omega$ is a better way to represent the adversary’s actual workload than the leakage from $\Omega$ 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, we will stick to Shannon entropy.) The relevant quantity in the POK scenario is the Rényi entropy $H_2$, which features in the 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 $I(X; \Omega)$ to the collisions in $\Omega$.

---

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

4Most biometrics cannot be kept secret, since it is possible to measure them surreptitiously.
Lemma 6 Consider the algorithm BioSCOM.Enroll or FESCOM.Enroll. Let \( t(W, \Omega) \) denote the number of entries in \( \Omega \) equal to \( W \), i.e. \( t(W, \Omega) = \{ j : \Omega_j = W \} \). Then

\[
H(X|\Omega) = H(X|W) + \log m - E_{\omega} \log t(w, \omega).
\]

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

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

\[
H(X|\Omega) \geq H(X|W) + \log m - \frac{m - 1}{\ln 2} E_{\omega} q_w.
\]

Proof: We start from Lemma 6. We write \( t(w, \omega) = 1 + u(w, \omega) \) and use \( \ln(1 + u) \leq u \). This gives \( E_{\omega} \log t(w, \omega) \leq \frac{1}{m} E_{\omega} u(w, \omega) \). 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 \( E_{\omega} u(w, \omega) = E_{\omega} [(m - 1) q_w] \). □ The probability \( q_w \) is typically of the order \( 2^{-n} \) if \( X \) is not too strangely distributed. Hence the last term in (10) is a small correction term if \( m < 2^{-n} \). 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 \( H(X|\Omega) \) can be bounded from below as

\[
H(X|\Omega) \geq H(X) - \frac{2^{-n-k} - 1}{\ln 2}
\]

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 \( \delta > 0 \). Then

\[
E_{\omega} \log t = E_{\omega} \log [m q_w] + E_{\omega} \log [1 + \frac{1 - q_w}{m q_w}]
\]

\[
+ E_{\omega} \log (1 + \frac{\delta}{1 + (m - 1) q_w}).
\]

Next we use \( \ln(1 + z) \leq z \) twice, and \( E_{\omega} \delta = 0 \), to get

\[
E_{\omega} \log t \leq \log m + E_{\omega} \log q_w + \frac{1}{\ln 2} E_{\omega} \frac{1 - q_w}{m q_w}
\]

\[
= \log m - H(W) + \frac{1}{m \ln 2} \left(-1 + \sum_z q_z\right)
\]

\[
= \log m - H(W) + \frac{2^{-n-k} - 1}{m \ln 2}.
\]

We substitute (13) into Lemma 6 and use \( H(X|W) + H(W) = H(X) \). □

In the limit \( m > 2^{-n} \), the \( \frac{1}{m} \) term in Theorem 2 is a small correction term; we see that \( \Omega \) then hardly leaks anything about \( X \), as we would expect intuitively, since \( \Omega \) 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 \( H_2(X|\Omega) \) that is useful for large \( m \). We observe that for the adversary each entry in \( \omega \) 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 \( \omega_j \), with equal probability \( 1/m \). (Remember that the fake helper data are drawn from the same distribution as \( Y \).) This gives

\[
p_{x|\omega} = \frac{1}{m} \sum_{j=1}^m p_{x|\omega_j}.
\]

On (14) we obtain the following result.

Theorem 3 Consider the algorithm BioSCOM.Enroll or FESCOM.Enroll. The conditional Rényi entropy \( H_2(X|\Omega) \) can be bounded from below as

\[
H_2(X|\Omega) \geq H_2(X) - \frac{1}{m \ln 2} \left[ \frac{E_{\omega} \sum_x p_x^2}{\sum_x p_x^2} - 1 \right].
\]

Proof:

\[
H_2(X|\Omega) = -2 \log E_{\omega} \sqrt{\sum_x p_x^2|\omega}
\]

\[
\geq -2 \log E_{\omega} \sum_x p_x^2|\omega
\]

\[
= \log E_{\omega} \sum_x \chi_{a,b}(\omega|\omega) p_x^2|\omega
\]

\[
= \log \left[ \frac{1}{m} E_{\omega} p_x^2|\omega + \frac{1}{m} \frac{1}{p_x^2} \right]
\]

\[
= \log \left( \sum_x p_x^2 \right) (1 + \frac{E_{\omega} \sum_x p_x^2|\omega - \sum_x p_x^2}}{m \sum_x p_x^2})
\]

\[
= H_2(X) - \log (1 + \frac{E_{\omega} \sum_x p_x^2|\omega} - \sum_x p_x^2}{m \sum_x p_x^2}).
\]

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 \( 2^{-n}/m \), i.e. the same order of magnitude as the \( \frac{1}{m} \) term in Theorem 2.

When spammed helper data \( \Omega \) is used instead of \( W \), the entropy \( H_2(X|W) \) in the extractable randomness formula (3) can be replaced by the (much) larger number \( H_2(X|\Omega) \).

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) $\beta$.

Example 1. Consider a uniform biometric $X$ with $n = 127$ and $\beta = 2.2\%$. (Such a low BER can be obtained by pre-processing, 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 $h(X|\Omega) = k = 50$ bits. In order to gain $\delta$ bits of security we need $m \approx 2^{\delta+1}$, which requires a storage space of $2^{\delta+1}(n - k) = 2^{\delta+5.5}$ 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 $\text{Syn}X$ 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 $\delta$ extra bits of security, the required storage is $2^{\delta+1}(n - k)$ bits $\approx 2^{10-3.3}$ KB. The column weight of the parity check matrix is 3, which means that each bit flip in $X'$ causes 3 bit flips in $\text{Syn}X'$. We have $d_H(\text{Syn}X', \text{Syn}X) \leq 3 \cdot d_H(X', X)$. 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 $\text{Syn}X'$. The Hamming distance between $\text{Syn}X$ (or $\text{Syn}X'$) 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 $\approx 161$ bit flips in the syndrome to distinguish between the real helper data and the fakes. The probability $\Pr[d_H(\text{Syn}X', \text{Syn}X) > 161]$ is less than $1.6 \cdot 10^{-4}$, while The probability that a random 408-bit string is removed from $\text{Syn}X$ (or $\text{Syn}X'$) by fewer than 161 bit flips is $1 \cdot 10^{-5}$. Hence, even if $\Omega$ 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 $2^{15-0.03}$ KB. The number of bit flips in $\text{Syn}X'$ has average $\leq 1296$ and standard deviation $\leq 60.6$, whereas a random 4000-bit string has average Hamming distance 2000 with standard deviation 31.6. We have $\Pr[d_H(\text{random string}, \text{Syn}X) < 1500] = 1 \cdot 10^{-6}$ and $\Pr[d_H(\text{Syn}X', \text{Syn}X) > 1590] \leq 1 \cdot 10^{-6}$, again a clear separation between the real entry and the fake ones.

8 Generative SCOM

The list $\Omega$ 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 $\Omega$ on the fly. This changes the security-storage tradeoff into a security-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* $\Omega$ than to fetch it from nonvolatile memory.

8.1 Construction

The construction relies on a fast PRNG $\gamma$ that generates $(n - k)$-bit strings, given a seed $S$. The $i$’th string derived from $S$ is denoted as $\gamma_i(S)$, i.e. $\gamma_0(S), \ldots, \gamma_N(S)$ on the Hamming distance to distinguish between likely candidates (small $d_H$) and unlikely candidates (large $d_H$). 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 $W = \text{Syn}X$.
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 $F = B' \oplus \text{Syn}X'$.
5. For $r = 1$ to $N$.
6. For $i = 1$ to $m$.
   (a) $\Delta_i = F \oplus \gamma_i(S')$.
   (b) If $\text{Hamw}(\Delta_i) \notin \{\theta_{i-1}, \ldots, \theta_i - 1\}$ then Next i.
   (c) Try to compute $E_i = S\text{Dec} \Delta_i$.
   If the decoding fails then Next i.
   (d) $X_i = X' \oplus E_i$.
   (e) If $f(m||S'||B'||X_i) = G'$ then Set Auth=True.
7. Next i.
8. If Auth=False then Next r.

- In GenSCOM.Verify, the mask $B'$ is added to $\text{Syn}X'$ 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 $d_H(W^{\text{fake}}, \text{Syn}X')$ is practically the same as the distribution of $d_H(W^{\text{fake}}, \text{Syn}X)$ 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 $\Omega$ is taken by the generated sequence $\Omega^{gen} = (B \oplus \gamma_i(S))_{i=1}^m$, with $\Omega^{gen}_{\beta}$ =
Syn X. The quantity of interest for the security is $H(X|SB)$, since $S$ and $B$ are stored as public data.

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

$$H(X|SB) = H(X|\Omega^{gen}).$$

**Proof:** We have $H(X|BS) = H(X|\Omega^{gen}S)$ since $\Omega^{gen}S$ together contain the same information as $B, S$. Finally we note that if $\Omega^{gen}$ 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

$$H(X|SB) \geq H(X|W) + \log m - \frac{m-1}{2^{n-k} \ln 2} - \left(1 - \frac{1}{m}\right) \left[ D(W||U) + D(U||W) \right].$$

**Proof:** We start from Lemma 7 and use Lemma 4, replacing $\Omega \rightarrow \Omega^{gen}$. We have $H(Z) = \log m$. For the collision penalty we get $H(Z|W|\Omega^{gen}) = E_{w_0} \log(1 + u) \leq E_{w_0} u / \ln 2$ just as in the proof of Theorem 1. Now, however, the entries are uniformly generated instead of having distribution $q_w$. This gives $E_{w_0} u = (m-1)/2^{n-k}$. Finally, for the distribution mismatch penalty $I(Z;\Omega^{gen})$ we use Lemma 5 with $W_f = U$ 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 $\Omega$. 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 $\Omega$ 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 $\text{Syn} X$ 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 $I(Z;\Omega)$ (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 \( \varphi_w = \Pr[W_{\text{fake}} = w] \).

\[
I(Z; \Omega) = -E_{w,z} \log \Pr[\Omega = \omega | Z = z] \Pr[Z = z] \\
= -E_{w,z} \log \Pr[\Omega = \omega | \Omega = \omega | Z = z] \\
= -E_{w,z} \log E_w \Pr[\Omega = \omega | Z = a] \\
= -E_{w,z} \log E_w \frac{q_{a,w} \varphi_w}{q_{a,w} \varphi_w} \\
= -E_{w,z} \pi_a \sum \frac{q_{a,w} \log \varphi_w - \log \varphi_w}{q_{a,w} \varphi_w} \\
= -E_{w,z} \sum \frac{\pi_a}{q_{a,w}} \sum q_{a,w} \varphi_w \log \frac{q_{a,w} \varphi_w}{q_{a,w} \varphi_w} \\
= E_z \left( 1 - \pi_z \right) \left[ D(W || W_{\text{fake}}) + D(W_{\text{fake}} || W) \right].
\]

In (25) we used the fact that \( \Pr[\Omega = \omega | Z = z] \) and \( \Pr[\Omega = \omega | Z = a] \) all the components \( \ell \neq a, z \) have \( \Pr[\Omega = \omega] = \Pr[W_{\text{fake}} = \omega] = \varphi_w \), which leads to cancellation in the fraction. In (26) we used Jensen’s inequality to write \( \log E_z \leq E_z \log \). In (27) we wrote out \( E_z \), using the fact that only the components \( \varphi_w \) and \( \omega_a \) are relevant, and that \( \omega_a \) is the actual helper data \( W \) while \( \omega_a \) is fake. In (28) we used the definition of the KL distance.

References