Fuzzy Extractors and Cryptography, or How to Use Your Fingerprints

Yevgeniy Dodis* Leonid Reyzin† Adam Smith‡

November 10, 2003

Abstract

We provide formal definitions and efficient secure techniques for

- turning biometric information into keys usable for *any* cryptographic application, and
- reliably and securely authenticating biometric data.

Our techniques apply not just to biometric information, but to any keying material that, unlike traditional cryptographic keys, is (1) not reproducible precisely and (2) not distributed uniformly. We propose two primitives: a *fuzzy extractor* extracts nearly uniform randomness R from its biometric input; the extraction is error-tolerant in the sense that R will be the same even if the input changes, as long as it remains reasonably close to the original. Thus, R can be used as a key in any cryptographic application. A *fuzzy fingerprint* produces public information about its biometric input w that does not reveal w, and yet allows exact recovery of w given another value that is close to w. Thus, it can be used to reliably reproduce error-prone biometric inputs without incurring the security risk inherent in storing them.

In addition to formally introducing our new primitives, we provide nearly optimal constructions of both primitives for various measures of "closeness" of input data, such as Hamming distance, edit distance, and set difference.

1 Introduction

Cryptography traditionally relies on uniformly distributed random strings for its secrets. Reality, however, makes it difficult to create, store, and reliably retrieve such strings. Strings that are neither uniformly random nor reliably reproducible seem to be more plentiful. For example, a random person's fingerprint or iris scan is clearly not a uniform random string, nor does it get reproduced precisely each time it is measured. Similarly, a long pass-phrase (or answers to 15 questions [11] or a list of favorite movies [16]) is not uniformly random and is difficult to remember for a human user. This work is about using such nonuniform and unreliable secrets in cryptographic applications. Our approach is rigorous and general, and our results have both theoretical and practical value.

To illustrate the use of random strings on a simple example, let us consider the task of password authentication. A user Alice has a password w and wants to gain access to her account. A trusted server stores some information $y = f(w)$ about the password. When Alice enters w, the server lets Alice in only if $f(w) = y$. In this simple application, we assume that it is safe for Alice to enter the password for the verification. However, the server's long-term storage is not assumed to be secure (e.g., y is stored in a publicly readable `/etc/passwd` file in UNIX [22]).

*`dodis@cs.nyu.edu`. New York University, Department of Computer Science, 251 Mercer St., New York, NY 10012 USA.
†`reyzin@cs.bu.edu`. Boston University, Department of Computer Science, 111 Cummington St., Boston MA 02215 USA.
‡`asmith@theory.lcs.mit.edu`. MIT, Laboratory for Computer Science, 200 Technology Sq., Cambridge, MA 02139 USA.

The goal, then, is to design an efficient f that is hard to invert (i.e., given y it is hard to find w' s.t. $f(w') = y$), so that no one can figure out Alice's password from y. Recall that such functions f are called *one-way functions*.

Unfortunately, the solution above has several problems when used with passwords w available in real life. First, the definition of a one-way function assumes that w is *truly uniform*, and guarantees nothing if this is not the case. However, human-generated and biometric passwords are far from uniform, although they do have some unpredictability in them. Second, Alice has to reproduce her password *exactly* each time she authenticates herself. This restriction severely limits the kinds of passwords that can be used. Indeed, a human can precisely memorize and reliably type in only relatively short passwords, which do not provide an adequate level of security. Greater levels of security are achieved by longer human-generated and biometric passwords, such as pass-phrases, answers to questionnaires, handwritten signatures, fingerprints, retina scans, voice commands, and other values selected by humans or provided by nature, possibly in combination (see [14, 10] for surveys). However, two biometric readings are rarely identical, even though they are likely to be close; similarly, humans are unlikely to precisely remember their answers to multiple question from time to time, though such answers will likely be similar. In other words, the ability to tolerate a (limited) number of errors in the password while retaining security is crucial if we are to obtain greater security than provided by typical user-chosen short passwords.

The password authentication described above is just one example of a cryptographic application where the issues of nonuniformity and error tolerance naturally come up. Other examples include any cryptographic application, such as encryption, signatures, or identification, where the secret key comes in the form of "biometric" data.

OUR DEFINITIONS. We propose two primitives, termed *fuzzy fingerprint* and *fuzzy extractor*.

A fuzzy fingerprint addresses the problem of error tolerance. It is a (probabilistic) function outputting a public value v about its biometric input w, that, while revealing little about w, allows its exact reconstruction from any other input w' that is sufficiently close. The price for this error tolerance is that the application will have to work with a lower level of entropy of the input, since publishing v effectively reduces the entropy of w. However, in a good fuzzy fingerpint, this reduction will be small, and w will still have enough entropy to be useful, even if the adversary knows v. A fuzzy fingerprint, however, does not address nonuniformity of inputs.

A fuzzy extractor addresses both error tolerance and nonuniformity. It reliably extracts a uniformly random string R from its biometric input w in an error-tolerant way. If the input changes but remains close, the extracted R remains the same. To assist in recovering R from w', a fuzzy extractor outputs a public string P (much like a fuzzy fingerprint outputs v to assist in recovering w). However, R remains uniformly random even given P.

Our approach is general: our primitives can be naturally combined with *any* cryptographic system. Indeed, R extracted from w by a fuzzy extractor can be used as a key in any cryptographic application, but, unlike traditional keys, need not be stored (because it can be recovered from any w' that is close to w). We define our primitives to be *information-theoretically* secure, thus allowing them to be used in combination with any cryptographic system without additional assumptions (however, the cryptographic application itself will typically have computational, rather than information-theoretic, security).

For a concrete example of how to use fuzzy extractors, in the password authentication case, the server can store $\langle P, f(R)\rangle$. When the user inputs w' close to w, the server recovers the actual R and checks if $f(R)$ matches what it stores. Similarly, R can be used for symmetric encryption, for generating a public-secret key pair, or any other application. Fuzzy fingerprints and extractors can thus be viewed as providing fuzzy key storage: they allow to recover the secret key (w or R) from a faulty reading w' of the password w, by using some public information (v or P). In particular, fuzzy extractors can viewed as error- and nonuniformity-tolerant secret key *key-encapsulation mechanisms* [28].

Because different biometric information has different error patterns, we do not assume any particular notion of closeness between w' and w. Rather, in defining our primitives, we simply assume that w comes from some metric space, and that w' is no more that a certain distance from w in that space. We only consider particular metrics when building concrete constructions, which are described below.

GENERAL RESULTS. Before proceeding to construct our primitives for concrete metrics, we make some observations about our definitions. We demonstrate that fuzzy extractors can be built out of fuzzy fingerprints by utilizing (the usual) strong randomness extractors [24], such as, for example, pairwise-independent hash functions. We also demonstrate that the existence of fuzzy fingerprints and fuzzy extractors over a particular metric space implies the existence of certain error-correcting codes in that space, thus producing lower bounds on the best parameters a fuzzy fingerprint and extractor can achieve. Finally, we define a notion of a *biometric embedding* of one metric space into another, and show that the existence of a fuzzy extractor in the target space implies, combined with a biometric embedding of the source into the target, the existence of a fuzzy extractor in the source space.

These general results help us in building and analyzing our constructions.

OUR CONSTRUCTIONS. We provide constructions of fuzzy fingerprints and extractors in three metrics: Hamming distance, set difference, and edit distance.

Hamming distance (i.e., the number of bit positions that differ between w and w') is perhaps the most natural metric to consider. We observe that the "fuzzy-commitment" construction of Juels and Wattenberg [15] based on error-correcting codes can be viewed as a (nearly optimal) fuzzy fingerprint. We then apply our general result to convert it into a nearly optimal fuzzy extractor. While our results on the Hamming distance essentially use previously known constructions, they serve as an important stepping stone for the rest of the work.

The set difference metric (i.e., size of the symmetric difference of two input sets w and w') comes up naturally whenever the biometric input is represented as a subset of features from a universe of possible features.[1] We demonstrate the existence of optimal (with respect to entropy loss) fuzzy fingerprints (and therefore also fuzzy extractors) for this metric. However, this result is mainly of theoretical interest, because (1) it relies on optimal constant-weight codes, which we do not know how construct and (2) it produces fingerprints of length proportional to the universe size. We then turn our attention to more efficient constructions for this metric, and provide two of them.

First, we observe that the "fuzzy vault" construction of Juels and Sudan [16] can viewed as a fuzzy fingerprint in this metric (and then converted to a fuzzy extractor using our general result). We provide a new, simpler analysis for this construction, which bounds the entropy lost from w given v. Our bound on the loss is quite high unless one makes the size of the output v very large. We then provide an improvement to the Juels-Sudan construction to reduce the entropy loss to near optimal, while keeping v short (essentially as long as w).

Second, we note that in the case of a small universe, a set can be simply encoded as its characteristic vector (1 if an element is in the set, 0 if it is not), and set difference becomes Hamming distance. However, the length of such a vector becomes unmanageable as the universe size grows. Nonetheless, we demonstrate that this approach can be made to work efficiently even for exponentially large universes. This involves a result that may be of independent interest: we show that BCH codes can be decoded in time polynomial in the *weight* of the received corrupted word (i.e., in *sublinear* time if the weight is small). The resulting fuzzy fingerprint scheme compares favorably to the modified Juels-Sudan construction: it has the same near-optimal entropy loss, while the public output v is even shorter (proportional to the number of errors tolerated, rather than the input length).

Finally, edit distance (i.e., the number of insertions and deletions needed to convert one string into the other) naturally comes up, for example, when the password is entered as a string, due to typing errors or mistakes made in handwriting recognition. We construct a biometric embedding from the edit metric into the set difference metric, and then apply our general result to show such an embedding yields a fuzzy extractor for edit distance, because we already have fuzzy extractors for set difference. We note that the edit metric is quite difficult to work with, and the existence of such an embedding is not a priori obvious: for example, low-distortion embeddings of the edit distance into the Hamming distance are unknown and seem hard [2]. It is the particular properties of biometric embeddings,

[1] A perhaps unexpected application of the set difference metric was explored in [16]: a user would like to encrypt a file (e.g., her phone number) using a small subset of values from a large universe (e.g., her favorite movies) in such a way that those and only those with a similar subset (e.g., similar taste in movies) can decrypt it.

as we define them, that help us construct this embedding.

RELATION TO PREVIOUS WORK. Since our work combines elements of error correction, randomness extraction and password authentication, there has been a lot of related work. Below we provide a few examples that we consider most relevant.

The need to deal with nonuniform and low-entropy passwords has long been realized in the security community, and many approaches have been proposed. For example, Kelsey et al [17] suggest using $f(w, r)$ in place of w for the password authentication scenario, where r is a public random "salt," to make a brute-force attacker's life harder. While practically useful, this approach does not add any entropy to the password, and does not formally address the needed properties of f. Another approach, more closely related to ours, is to add biometric features to the password. For example, Ellison et al. [9] propose asking the user a series of n personalized questions, and use these answers to encrypt the "actual" truly random secret R. A similar approach using user's keyboard dynamics (and, subsequently, voice [20, 21]) was proposed by Monrose et al [19]. Of course, this technique reduces the question to that of designing a secure "fuzzy encryption". While heuristics approaches were suggested in the above works (using various forms of Shamir's secret sharing [27]), no formal analysis was given. Additionally, error tolerance was addressed only by brute force search.

The most formal approach to error tolerance in biometrics was taken by Juels and Wattenberg [15] (for less formal solutions, see [8, 19, 9]), who provided a simple way to tolerate errors in *uniformly distributed* passwords.[2] Almost the same construction appeared implicitly in earlier, seemingly unrelated, literature on information reconciliation and privacy amplification (see, e.g., [3, 4, 7]). We further discuss the connections among these works and our work in Section 4.

Work on privacy amplification [3, 4], as well as work on de-randomization and hardness amplification [13, 24], also addressed the need to extract uniform randomness from a random variable about which some information has been leaked. In addition, work on (ordinary, not fuzzy) extractors is relevant to ours (e.g., [26, 23]), though for our purposes simple pairwise hashing is a sufficiently good extractor [3, 13][3].

It is also important to note the recent work of Juels and Sudan [16], who provide the first construction for a metric other than Hamming: they construct a "fuzzy vault" scheme for the set difference metric. The main difference is that [16] lacks a cryptographically strong definition of the object constructed. In particular, their construction leaks a significant amount of information about their analog of R, even though it leaves the adversary with provably "many valid choices" for R. In retrospect, their notion can be viewed as an (information-theoretically) one-way function, rather than semantically-secure key encapsulation mechanism, like the one considered in this work. Nonetheless, their informal notion is very closely related to our fuzzy fingerprints, and we improve their construction in Section 5.

EXTENSIONS. We can relax the error correction properties of fuzzy fingerprints/extractors to allow *list decoding*: instead of outputting one correct secret, we can output a short list of secrets, one of which is correct. For many applications (e.g., password authentication), this is sufficient, while the advantage is that we can possibly tolerate many more errors in the password. Not surprisingly, by using list-decodable codes (see [12] and the references therein) in our constructions, we can achieve this relaxation and considerably improve our error tolerance. Other similar extensions would be to allow small error probability in error-correction, to ensure correction of only *average-case* errors, or to consider nonbinary alphabets. Again, many of our results will extend to these settings. Finally, an interesting new direction is to consider other metrics not considered in this work.

[2] [11] extended this solution for better practical use, but it still works only for uniformly chosen passwords.

[3] This is so because optimizing the seed length — the major source of problems in extractor constructions — will not be crucially important in most of our applications, as long as one extracts (nearly) all the min-entropy from the source.

2 Preliminaries

Unless explicitly stated otherwise, all logarithms below are base 2.

ENTROPY. The *min-entropy* of a random variable A is $\mathbf{H}_\infty(A) = -\log(\max_a \Pr(A = a))$. For a pair of (possibly correlated) random variables A, B, a conventional notion of "average min-entropy" of A given B would be $\mathbf{H}_\infty(A \mid B) = \mathbb{E}_{b \leftarrow B}[\mathbf{H}_\infty(A \mid B = b)]$. However, for the purposes of this paper, the following slightly modified notion will be more robust: we let $\tilde{\mathbf{H}}_\infty(A \mid B) = -\log\left(\mathbb{E}_{b \leftarrow B}\left[2^{-\mathbf{H}_\infty(A|B=b)}\right]\right)$. Namely, we define *average min-entropy* of A given B to be the logarithm of the average probability of the most likely value of A given B. This definition is the right one to use when one is interested in the statistical difference from uniform, as becomes clear, e.g., in Lemma 3.2. One can easily verify that if B is an ℓ-bit string, then $\tilde{\mathbf{H}}_\infty(A \mid B) \geq \mathbf{H}_\infty(A) - \ell$.

STRONG EXTRACTORS. The *statistical distance between* two probability distributions A and B is $\mathbf{SD}(A, B) = \frac{1}{2}\sum_v |\Pr(A = v) - \Pr(B = v)|$. We can now define *strong randomness extractors* [24].

Definition 1. *An efficient* (n, m', ℓ, ϵ)-strong extractor *is a polynomial time probabilistic function* $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^\ell$ *such that for all min-entropy* m' *distributions* W*, we have* $\mathbf{SD}(\langle \mathsf{Ext}(W; X), X\rangle, \langle U_\ell, X\rangle) \leq \epsilon$*, where* $\mathsf{Ext}(W; X)$ *stands for applying* Ext *to* W *using (uniformly distributed) randomness* X*.*

Strong extractors can extract at most $\ell = m' - 2\log(1/\epsilon) + O(1)$ nearly random bits [25]. Many constructions match this bound (see Shaltiels' survey [26] for references). Extractor constructions are often complex since they seek to minimize the length of the seed X. For our purposes, the length of X will be less important, so 2-wise independent hash functions will already give us optimal $\ell = m' - 2\log(1/\epsilon)$ [3, 13].

METRIC SPACES. A metric space is a set $\mathcal{M}$ with a distance function $\mathsf{dis} : \mathcal{M} \times \mathcal{M} \to \mathbb{R}^+ = [0, \infty)$ which obeys various natural properties. In this work, $\mathcal{M}$ will always be a finite set, and the distance function will only take on integer values. The size of the $\mathcal{M}$ will always be denoted $N = |\mathcal{M}|$. We will assume that any point in $\mathcal{M}$ can be naturally represented as a binary string of appropriate length $O(\log N)$.

We will concentrate on the following metrics. (1) *Hamming metric*. Here $\mathcal{M} = \mathcal{F}^n$ over some alphabet $\mathcal{F}$ (we will mainly use $\mathcal{F} = \{0,1\}$), and $\mathsf{dis}(w, w')$ is the number of positions in which they differ. (2) *Set Difference metric*. Here $\mathcal{M}$ consists of all s-element subsets in a universe $\mathcal{U} = [n] = \{1, ..., n\}$. The distance between two sets A, B is the number of points in A that are not in B. Since A and B have the same size, the distance is half of the size of their symmetric difference: $\mathsf{dis}(A, B) = \frac{1}{2}|A \triangle B|$. (3) *Edit metric*. Here again $\mathcal{M} = \mathcal{F}^n$, but the distance between w and w' is defined to be one half of the smallest number of character insertions and deletions needed to transform w into w'.

As already mentioned, all three metrics seem natural for biometric data.

CODING. Since we want to achieve error tolerance in various metric spaces, we will use *error-correcting codes* in the corresponding metric space $\mathcal{M}$. A code C is a subset $\{w_1, \ldots, w_K\}$ of K elements of $\mathcal{M}$ (for efficiency purposes, we want the map from i to w_i to be polynomial-time). The *minimum distance* of C is the smallest $d > 0$ such that for all $i \neq j$ we have $\mathsf{dis}(w_i, w_j) \geq d$. In our case of integer metrics, this means that one can detect up to $(d-1)$ "errors" in any codeword. The *error-correcting distance* of C is the largest number $t > 0$ such that for every $w \in \mathcal{M}$ there exists at most one codeword w_i in the ball of radius t around w: $\mathsf{dis}(w, w_i) \leq t$ for at most one i. Clearly, for integer metrics we have $d = 2t + 1$. Since error correction will be more important in our applications, we denote the corresponding codes by $(\mathcal{M}, K, t)$-codes. For the Hamming and the edit metrics on strings of length n over some alphabet $\mathcal{F}$, we will sometimes call $k = \log_{|F|} K$ the *dimension* on the code, and denote the code itself as an $[n, k, d = 2t + 1]$-code, following the standard notation in the literature.

3 Definitions and General Lemmas

Let $\mathcal{M}$ be a metric space on N points with distance function dis.

Definition 2. *An* $(\mathcal{M}, m, m', t)$-fuzzy fingerprint *is a randomized map* $\mathsf{FF} : \mathcal{M} \to \{0,1\}^*$ *with the following properties.*

1. *There exists a deterministic recovery function* Rec *allowing to recover* w *from its fingerprint* $\mathsf{FF}(w)$ *and any vector* w' *close to* w*: for all* $w, w' \in \mathcal{M}$ *satisfying* $\mathsf{dis}(w, w') \le t$*, we have* $\mathsf{Rec}(w', \mathsf{FF}(w)) = w$*.*
2. *For all random variables* W *over* $\mathcal{M}$ *with min-entropy* m*, the average min-entropy of* W *given* $\mathsf{FF}(W)$ *is at least* m'*. That is,* $\tilde{\mathbf{H}}_\infty(W \mid \mathsf{FF}(W)) \ge m'$*.*

The fuzzy fingerprint is efficient *if* FF *and* Rec *run in time polynomial in the representation size of a point in* $\mathcal{M}$*. We denote the random output of* FF *by* $\mathsf{FF}(W)$*, or by* $\mathsf{FF}(W; X)$ *when we wish to make the randomness explicit.*

We will have several examples of fuzzy fingerprints when we deal with specific metrics. The quantity $m - m'$ is called the *entropy loss* of a fuzzy fingerprint. All of our proofs in fact bound $m - m'$, and the same bound typically holds for all values of m.

Definition 3. *An* $(\mathcal{M}, m, \ell, t, \epsilon)$ fuzzy extractor *is a given by two procedures* $(\mathsf{Gen}, \mathsf{Rep})$.

1. Gen *is a probabilistic generation procedure, which on input* $w \in \mathcal{M}$ *outputs an "extracted" string* $R \in \{0,1\}^\ell$ *and a public string* P*. We require that for any distribution* W *on* $\mathcal{M}$ *of min-entropy* m*, if* $\langle R, P\rangle \leftarrow \mathsf{Gen}(W)$*, then we have* $\mathbf{SD}\left(\langle R, P\rangle, \langle U_\ell, P\rangle\right) \le \epsilon$*, where* U_ℓ *is the uniform distribution on* $\{0,1\}^\ell$ *sampled independently from* P*.*
2. Rep *is a deterministic reproduction procedure allowing to recover* R *from the corresponding public string* P *and any vector* w' *close to* w*: for all* $w, w' \in \mathcal{M}$ *satisfying* $\mathsf{dis}(w, w') \le t$*, if* $\langle R, P\rangle \leftarrow \mathsf{Gen}(w)$*, then we have* $\mathsf{Rep}(w', p) = R$*.*

The fuzzy extractor is efficient *if* Gen *and* Rep *run in time polynomial in the representation size of a point in* $\mathcal{M}$*.*

In other words, fuzzy extractors allow one to extract some randomness R from W and then successfully reproduce R from any string W' that is close to W. The reproduction is done with the help of the public string P produced during the initial extraction; yet R looks truly random even given P. To justify our terminology, notice that strong extractors (as defined in Section 2) can indeed be seen as "nonfuzzy" analogs of fuzzy extractors, corresponding to $t = 0$, $P = X$ (and $\mathcal{M} = \{0,1\}^n$).

CONSTRUCTION OF FUZZY EXTRACTORS FROM FUZZY FINGERPRINTS. Not surprisingly, fuzzy fingerprints come up very handy in constructing fuzzy extractors. Specifically, we construct fuzzy extractors from fuzzy fingerprints and strong extractors. For that, we assume that one can naturally represent a point w in $\mathcal{M}$ using n bits. The strong extractor we use is the standard pairwise-independent hashing construction, which has (optimal) entropy loss $2\log\left(\frac{1}{\epsilon}\right)$. The result follows from the "left-over hash" (a.k.a. "privacy amplification") lemmas of [13, 4].

Lemma 3.1 (Using Hash Functions for Extraction). *Assume* FF *is a* $(\mathcal{M}, m, m', t)$*-fuzzy fingerprint with recovery procedure* Rec*, and let* Ext *be the* (n, m', ℓ, ϵ)*-strong extractor based on pairwise-independent hashing (in particular,* $\ell = m' - 2\log\left(\frac{1}{\epsilon}\right)$*). Then the following* $(\mathsf{Gen}, \mathsf{Rep})$ *is a* $(\mathcal{M}, m, \ell, t, \epsilon)$*-fuzzy extractor:*

- $\mathsf{Gen}(W; X_1, X_2)$*: set* $P = \langle \mathsf{FF}(W; X_1), X_2\rangle$*,* $R = \mathsf{Ext}(W; X_2)$*, and output* $\langle R, P\rangle$*.*
- $\mathsf{Rep}(W', \langle V, X_2\rangle)$*: recover* $W = \mathsf{Rec}(W', V)$ *and output* $R = \mathsf{Ext}(W; X_2)$*.*

Proof. Lemma 3.1 follows directly from the intermediate result below (Lemma 3.2), which explains our choice of the measure $\tilde{\mathbf{H}}_\infty(A|B)$ for the average min-entropy. Lemma 3.2 says that pairwise independent hashing extracts randomness from the random variable A as if the min-entropy of A given $B = b$ were always at least $\tilde{\mathbf{H}}_\infty(A|B)$ (rather than having the inequality hold on average). □

Interestingly, the above lemma, combined with fuzzy fingerprints, will often produce nearly optimal fuzzy extractors.

Lemma 3.2. *If A, B are random variables such that $A \in \{0,1\}^n$ and $\tilde{\mathbf{H}}_\infty(A|B) \geq m'$, and H is a random pairwise independent hash function from n bits to ℓ bits, then $\mathbf{SD}\left(\langle B, H, H(A)\rangle\ ,\ \langle B, H, U_\ell\rangle\right) \leq \epsilon$ as long as $\ell \leq m' - 2\log\left(\frac{1}{\epsilon}\right)$.*

Proof. The particular extractor we chose has a smooth tradeoff between the entropy of the input and the quality of the output. For any random variable X, the left-over hash/privacy amplification lemma [3, 13, 4] states:

$$\mathbf{SD}\left(\langle H, H(X)\rangle\ ,\ \langle H, U_\ell\rangle\right) \leq \sqrt{2^{-\mathbf{H}_\infty(X)} 2^\ell}$$

In our setting we have a bound on the *expected* value of $2^{-\mathbf{H}_\infty(A|B=b)}$, namely $\mathbb{E}\left[2^{-\mathbf{H}_\infty A} \mid B)\right] \leq 2^{-m'}$. Using the fact that $\mathbb{E}\left[\sqrt{Z}\right] \leq \sqrt{\mathbb{E}\left[Z\right]}$, we get:

$$\mathbb{E}_b\left[\mathbf{SD}\left(\langle H, H(A|B=b)\rangle\ ,\ \langle H, U_\ell\rangle\right)\right] \leq \sqrt{2^{\ell - m'}}.$$

Now the distance of $\langle B, H, H(A)\rangle$ from $\langle B, H, U_\ell\rangle$ is the average over values of B of the distance of $\langle H, H(A)\rangle$ from $\langle H, U_\ell\rangle$. This average is exactly what was bounded above:

$$\mathbf{SD}\left(\langle B, H, H(A)\rangle, \langle P, U_\ell\rangle\right) = \mathbb{E}_B\left[\mathbf{SD}\left(\langle H, H(A)\rangle\ ,\ \langle H, U_\ell\rangle\right)\right] \leq \sqrt{2^{\ell - m'}}.$$

The extractor we use always has $\ell \leq m' - 2\log\left(\frac{1}{\epsilon}\right)$, and so the statistical difference is at most ϵ. □

Remark 1. One can prove an analogous form of Lemma 3.2 using any strong extractor. If the extractor does not have a convex tradeoff between the input entropy and the distance from uniform of the output, then one can instead use a high-probability bound on the min-entropy of the input (that is, if $\tilde{\mathbf{H}}_\infty(X) \geq m'$ then the event $\mathbf{H}_\infty(X) \geq m' - \log\left(\frac{1}{\epsilon}\right)$ happens with probability $1 - \epsilon$). The resulting reduction leads to fuzzy extractors with min-entropy loss $3\log\left(\frac{1}{\epsilon}\right)$.

CONSTRUCTIONS FROM BIOMETRIC EMBEDDINGS. We now introduce a general technique that allows one to build good fuzzy extractors in some metric space $\mathcal{M}_1$ from good fuzzy extractors in some other metric space $\mathcal{M}_2$. Below, we let $\mathsf{dis}(\cdot,\cdot)_i$ denote the distance function in $\mathcal{M}_i$. The technique is to *embed* $\mathcal{M}_1$ into $\mathcal{M}_2$ so as to "preserve" relevant parameters for fuzzy extraction.

Definition 4. *A function $f : \mathcal{M}_1 \to \mathcal{M}_2$ is called a (t_1, t_2, m_1, m_2)-biometric embedding if the following two conditions hold:*

- *for any $w_1, w_1' \in \mathcal{M}_1$ such that $\mathsf{dis}(w_1, w_1')_1 \leq t_1$, we have $\mathsf{dis}(f(w_1), f(w_2))_2 \leq t_2$.*
- *for any W_1 on $\mathcal{M}_1$ of min-entropy at least m_1, $f(W_1)$ has min-entropy at least m_2.*

The following lemma is immediate:

Lemma 3.3. *If f is (t_1, t_2, m_1, m_2)-biometric embedding of $\mathcal{M}_1$ into $\mathcal{M}_2$ and $(\mathsf{Gen}_1(\cdot), \mathsf{Rep}_1(\cdot,\cdot))$ is a $(\mathcal{M}_2, m_2, \ell, t_2, \epsilon)$-fuzzy extractor, then $(\mathsf{Gen}_1(f(\cdot)), \mathsf{Rep}_1(f(\cdot),\cdot))$ is a $(\mathcal{M}_1, m_1, \ell, t_1, \epsilon)$-fuzzy extractor.*

Notice that a similar result does not hold for fuzzy fingerprints, unless f is injective (and efficiently invertible).

We will see the utility of this particular notion of embedding (as opposed to previously defined notions) in Section 6.

4 Constructions for Hamming Distance

In this section we consider constructions for the space $\mathcal{M} = \{0,1\}^n$ under the Hamming distance metric.

THE CODE-OFFSET CONSTRUCTION. Juels and Wattenberg [15] considered a notion of "fuzzy commitment." [4] Given a binary $[n, k, 2t+1]$ error-correcting code C (not necessarily linear), they fuzzy-commit to X by publishing $W \oplus C(X)$. Their construction can be rephrased in our language to give a very simple construction of fuzzy fingerprints: for random $X \leftarrow \{0,1\}^k$, set

$$\mathsf{FF}(W; X) = W \oplus C(X)\,.$$

(Note that if W is uniform, this fuzzy fingerprint direcly yields a fuzzy extractor with $R = X$).

When the code C is linear, this is equivalent to revealing the syndrome of the input w, and so we do not need the randomness X. Namely, in this case we could have set $\mathsf{FF}(w) = \mathsf{syn}_C(w)$ (as mentioned in the introduction, this construction also appears implicitly in the information reconciliation literature, e.g. [3, 4, 7]: when Alice and Bob hold secret values which are very close in Hamming distance, one way to correct the differences with few bits of communication is for Alice to send to Bob the *syndrome* of her word w with respect to a good linear code.)

Since the syndrome of a k-dimensional linear code is $n-k$ bits long, it is clear that $\mathsf{FF}(w)$ leaks only $n-k$ bits about w. In fact, we show the same is true even for nonlinear codes.

Lemma 4.1. *For any $[n, k, 2t+1]$ code C and any m, FF above is a $(\mathcal{M}, m, m+k-n, t)$ fuzzy fingerprint. It is efficient if the code C allows decoding errors in polynomial time.*

Proof. Let D be the decoding procedure of our code C. Since D can correct up to t errors, if $v = w \oplus C(x)$ and $\mathsf{dis}(w, w') \le t$, then $D(w' \oplus v) = x$. Thus, we can set $\mathsf{Rec}(w', v) = v \oplus C(D(w' \oplus v))$.

Let A be the joint variable (X, W). Together, these have min-entropy $m + k$ when $\mathbf{H}_\infty(W) = m$. Since $\mathsf{FF}(W) \in \{0,1\}^n$, we have $\tilde{\mathbf{H}}_\infty(W, X \mid \mathsf{FF}(W)) \ge m + k - n$. Now given $\mathsf{FF}(W)$, W and X determine each other uniquely, and so $\tilde{\mathbf{H}}_\infty(W \mid \mathsf{FF}(W)) \ge m + k - n$ as well. □

In Appendix A, we present some generic lower bounds on fuzzy fingerprints and extractors. Let $A(n, d)$ denote the maximum number of codewords possible in a code of distance d in $\{0,1\}^n$. Then Lemma A.1 implies that the entropy loss of a fuzzy fingerprint for the Hamming metric is at least $n - \log A(n, 2t+1)$, when the input is uniform (that is, when $m = n$). This means that the code-offset construction above is optimal for the case of uniform inputs. Of course, we do not know the exact value of $A(n, d)$, never mind of efficiently decodable codes which meet the bound, for most settings of n and d. Nonetheless, the code-offset scheme gets as close to optimality as is possible in coding.

GETTING FUZZY EXTRACTORS. As a warm-up, consider the case when W is uniform ($m = n$) and look at the code-offset fingerprint construction: $V = W \oplus C(X)$. Setting $R = X$, $P = V$ and $\mathsf{Rep}(W', V) = D(V \oplus W')$, we clearly get an $(\mathcal{M}, n, k, t, 0)$ fuzzy extractor, since V is truly random when W is random, and therefore independent of X. In fact, this is exactly the usage proposed by Juels-Wattenberg, except they viewed the above fuzzy extractor as a way to use W to "fuzzy commit" to X, without revealing information about X.

Unfortunately, the above construction setting $R = X$ only works for uniform W, since otherwise V would leak information about X. However, by using the construction in Lemma 3.1, we get

Lemma 4.2. *Given any $[n, k, 2t+1]$ code C and any m, ϵ, we can get an $(\mathcal{M}, m, \ell, t, \epsilon)$ fuzzy extractor, where $\ell = m + k - n - 2\log(1/\epsilon)$. The recovery Rep is efficient if C allows decoding errors in polynomial time.*

[4] In their interpretation, one commits to X by picking a random W and publishing $\mathsf{FF}(W; X)$.

5 Constructions for Set Difference

Consider the collection of all sets of a particular size s in a universe $\mathcal{U} = [n] = \{1, ..., n\}$. The distance between two sets A, B is the number of points in A that are not in B. Since A and B have the same size, the distance is half of the size of their symmetric difference: $\frac{1}{2}\mathsf{dis}(A, B) = |A \triangle B|$. If A and B are viewed as n-bit characteristic vectors over $[n]$, this metric is the same as the Hamming metric (scaled by $1/2$). Thus, the set difference metric can be viewed as a restriction of the binary Hamming metric to all the strings with exactly s nonzero components. However, one typically assumes that n is much larger than s, so that representing a set by n bits is much less efficient than, say writing down a list of elements, which requires $(s \log n)$ bits.

LARGE VERSUS SMALL UNIVERSES. Most of this section studies situations where the universe size n is super-polynomial in the set size s. We call this the large universe setting. By contrast, the small universe setting refers to situations in which $n = poly(s)$. We want our various constructions to run in polynomial time and use polynomial storage space. Thus, the large universe setting is exactly the setting in which the n-bit string representation of a set becomes too large to be usable. We consider the small-universe setting first, since it appears simpler (Section 5.1). The remaining subsections consider large universes.

5.1 Small Universes

When the universe size is polynomial in s, there are a number of natural constructions. Perhaps the most direct one, given previous work, is the construction of Juels and Sudan [16]. Unfortunately, that scheme achieves relatively poor parameters (see Section 5.2).

We suggest two possible constructions: first, to represent sets as n-bit strings and use the constructions of the previous section (with the caveat that Hamming distance is off by a factor of 2 from set difference). The second construction, presented below, goes directly through codes for set difference, also called "constant-weight" codes.

In order to be able to compare the constructions, and for consistency with the coding theory literature, we will in fact work with the Hamming metric here. Thus, codes which correct any t errors in set difference will have minimum distance at least $4t + 1$.

FINGERPRINTS FOR TRANSITIVE METRIC SPACES. The code-offset construction suggests a general technique for building fuzzy fingerprints in other metric spaces, using any code and particular sets of permutations. A permutation π on a metric space $\mathcal{M}$ is an *isometry* if it preserves distances, i.e. $\mathsf{dis}(a, b) = \mathsf{dis}(\pi(a), \pi(b))$. A family of permutations $\Pi = \{\pi_i\}_{i \in \mathcal{I}}$ acts *transitively* on $\mathcal{M}$ if for any two elements $a, b \in \mathcal{M}$, there exists $\pi_i \in \Pi$ such that $\pi_i(a) = b$. Suppose we have a family Π of transitive isometries for $\mathcal{M}$ (we will call such $\mathcal{M}$ *transtive*). For example, in the Hamming space, the set of all shifts $\pi_x(w) = w \oplus x$ is such a family. Then a natural fingerprinting scheme, given a password $a \in \mathcal{M}$, is to pick a random element b from the code, pick a random permutation $\pi \in \Pi$ such that $\pi(a) = b$, and output $\mathsf{FF}(A) = \pi$. Given a potential password a' and the fingerprint π, we can find the closest codeword to $\pi(a')$ (call that element b'), and output $\pi^{-1}(b')$. If the code has high minimum distance and a' is sufficiently close to a, then b' will be exactly b, and we will recover a.

The entropy loss of this scheme will depend on several parameters, but if the code has many elements, and if π can be described using few bits, then counting entropies can yield an interesting bound. (For the scheme to be usable, we also need the operations on the code, as well as π and π^{-1}, to be implementable reasonably efficiently.)

FINGERPRINTING FOR SET DIFFERENCE. We illustrate this general approach for set difference. The family of permutations we use is simply the one induced by the set of all permutations on the universe $[n]$. Let $C \subseteq \{0, 1\}^n$ be any $[n, k, d]$ code (nonlinear) in which all words have weight exactly s, and view elements of the code as sets of size s. We obtain the following scheme, which produces a fingerprint of length $n \log n$:

Algorithm 1 (Permutation-based fingerprint). Input: a set $A \subseteq \mathcal{U} = [n]$ of size s.

1. Choose $B \subseteq [n]$ at random from the code C.
2. Choose a random permutation $\pi : [n] \to [n]$ such that $\pi(A) = B$:
   (That is, choose a random matching between A and B and a random matching between $[n] - A$ and $[n] - B$.)
3. Output $\mathsf{FF}(A) = \pi$ (say, by listing $\pi(1), ..., \pi(n)$).

Lemma 5.1. *Suppose that C is a $[n, k, d]$ constant-weight-s code (for Hamming distance), then:*

1. *If $d \geq 4t+1$, there is an algorithm $\mathsf{Rec}()$ such that $\mathsf{Rec}(A', \mathsf{FF}(A)) = A$ for any sets A, A' such that $\frac{1}{2}|A \triangle A'| \leq t$. The algorithm is efficient if C has an efficient decoding algorithm.*
2. *The left-over entropy is $\tilde{\mathbf{H}}_\infty(A \mid \mathsf{FF}(A)) \geq \mathbf{H}_\infty(A) + k - \log \binom{n}{s}$.*

Proof. (1) Given π and A', we can compute $B' = \pi^{-1}(A')$. The intersection of B and B' is the same size as $A \cap A'$, and so the Hamming distance between the characteristic vectors of B and B' is at most $2t$. Since the code has minimum distance $d \geq 4t + 1$, it can correct $2t$ errors, and so the closest codeword is $B = \pi^{-1}(A)$. All operations are $n \log n$-time except for (possibly) the random choice of B in the algorithm and the decoding.

(2) Let X be the randomness used by the fingerprinting algorithm. There are $s!$ possibilities for the matching from A to B and $(n - s)!$ possibilities for the matching from $[n] - A$ to $[n] - B$. Hence, the min-entropy of the pair (A, X) is $\mathbf{H}_\infty(A) + \log(s!(n - s)!)$. There are $n!$ possibilities for the fingerprint π, and so the average min-entropy of (A, X) given $\mathsf{FF}(A)$ is at least $\mathbf{H}_\infty(A) + \log(s!(n - s)!) - \log(n!) = \mathbf{H}_\infty(A) - \log \binom{n}{s}$. Given A and $\mathsf{FF}(A)$ we can recover X exactly, and so $\tilde{\mathbf{H}}_\infty(A \mid \mathsf{FF}(A))$ is the same as $\tilde{\mathbf{H}}_\infty(A, X \mid \mathsf{FF}(A))$. □

COMPARING THE HAMMING SCHEME WITH THE PERMUTATION SCHEME. In order to get a feeling for how the random permutation technique compares to simply using Hamming-based schemes directly, we recall some notation from the coding theory literature. Let $A(n, d, s)$ denote the maximum size of a binary code for which all codewords have weight exactly s. Here n is the length of the code and d is the minimum distance. Let $A(n, d)$ denote the maximum size of an (unrestricted) binary code of length n and minimum distance d. In all cases, we're interested in codes with minimum distance $d \geq 4t + 1$, since we want to correct t errors in the set difference metric.

The code-offset construction was shown to have entropy loss $n - \log A(n, d)$ if an optimal code is used; the random permutation scheme can have entropy loss $\log \binom{n}{s} - \log A(n, d, s)$ for an optimal code. The Bassalygo-Elias inequality (see [18]) shows that the bound on the random permutation scheme is always at least as good as the bound on the code offset scheme: $A(n, d) \cdot 2^{-n} \leq A(n, d, s) \cdot \binom{n}{s}^{-1}$. This implies that $n - \log A(n, d) \geq \log \binom{n}{s} - \log A(n, d, s)$. Moreover, standard packing arguments give better constructions of constant-weight codes than they do of ordinary codes. [5] In fact, the random permutations scheme is optimal for this metric, just as the code-offset scheme is optimal for the Hamming metric: Lemma A.1 shows that the min-entropy loss of a fuzzy fingerprint must be at least $\log \binom{n}{s} - \log A(n, d, s)$, in the case of a uniform secret set A. Thus in principle, it is better to use the random permutation scheme. Nonetheless, there are caveats. First, we do not know of *explicitly* constructed constant-weight codes that beat the Elias-Bassalygo inequality and would thus lead to better entropy loss for the random permutation scheme than for the Hamming scheme (see [6] for more on construcionts of constant-weight codes and [1] for upper bounds). Second, much more is known about efficient implementation of decoding for ordinary codes than for constant-weight codes; for example, one can find off-the-shelf hardware and software for decoding many binary codes. In practice, the Hamming-based scheme is likely to be more useful.

[5] This comes from the fact that the intersection of a ball of radius d with the set of all words of weight s is much smaller than the ball of radius d itself.

5.2 Modifying the Construction of Juels and Sudan

We now turn to the large universe setting, where n is super-polynomial in s. Juels and Sudan [16] proposed a fuzzy fingerprinting protocol for the set difference metric (called a "fuzzy vault" in that paper). They assume for simplicity that $n = |\mathcal{U}|$ is a prime power and work over the field $\mathcal{F} = GF(n)$. On input set A, the fingerprint they produce is a set of r pairs of points (x_i, y_i) in $\mathcal{F}$, with $s < r \leq n$.

Algorithm 2 (Juels-Sudan Fuzzy Fingerprint). Input: a set $A \subseteq \mathcal{U}$.

1. Choose $p()$ at random from the set of polynomials of degree at most $k = s - 2t - 1$ over $\mathcal{F}$. Write $A = \{x_1, ..., x_s\}$, and let $y_i = p(x_i)$ for $i = 1, ..., s$.
2. Choose $r - s$ distinct points $x_{s+1}, ..., x_r$ at random from $\mathcal{F} - A$.
3. For $i = s + 1, ..., r$, choose $y_i \in \mathcal{F}$ at random such that $y_i \neq p(x_i)$.
4. Output $\mathsf{FF}(A) = \{(x_1, y_1), ..., (x_r, y_r)\}$ (in lexicographic order of x_i).

The parameter r dictates the amount of storage necessary, one on hand, and also the security of the scheme (that is, for $r = s$ the scheme leaks all information and for larger and larger r there is less information about A). Juels and Sudan actually propose two analyses for the scheme. First, they analyze the case where the secret A is distributed uniformly over all subsets of size s. Second, they provide an analysis of a nonuniform password distribution, but only for the case $r = n$ (that is, their analysis only applies in the small universe setting, where $\Omega(n)$ storage is acceptable). Here we give a simpler analysis which handles nonuniformity and any $r \leq n$. We get the same results for a broader set of parameters.

Lemma 5.2. *The entropy loss of the Juels-Sudan scheme* $\mathsf{FF}()$ *above is at most* $2t \log n + \log \binom{n}{r} - \log \binom{n-s}{r-s}$.

Proof. As for the code-offset, we can simply count entropies. Let X denote the random bits used by the algorithm to generate $\mathsf{FF}(A)$. Choosing the polynomial p requires $s - 2t$ random choices from $\mathcal{F}$. The choice of the remaining x_i's requires $\log \binom{n-s}{r-s}$ bits, and choosing the $y_i's$ requires $r - s$ random choices from $\mathcal{F}$ (we will ignore the difference between $\mathcal{F}$ and $\mathcal{F} - \{x_i\}$ here since it doesn't affect the result significantly). The min-entropy of the pair A, X is thus $\tilde{\mathbf{H}}_\infty(A, X) = \tilde{\mathbf{H}}_\infty(A) + (r - 2t)\log(n) + \log \binom{n-s}{r-s}$. The output can be described in $\log \left(\binom{n}{r} n^r\right)$ bits, and hence we get that $\tilde{\mathbf{H}}_\infty(A, X \mid \mathsf{FF}(A)) = \tilde{\mathbf{H}}_\infty(A) - 2t \log n + \log \binom{n-s}{r-s} - \log \binom{n}{r}$. Finally, note that X is entirely determined by A and $\mathsf{FF}(A)$, so the entropy of A, X given $\mathsf{FF}(A)$ is the same as the entropy of A given $\mathsf{FF}(A)$. □

In the large universe setting, we will have $r \ll n$ (since we wish to have storage polynomial in s). In that setting, the bound on the entropy loss of the Juels-Sudan scheme is in fact very large. We can rewrite the entropy loss as $2t \log n - \log \binom{r}{s} + \log \binom{n}{s}$, using the identity $\binom{n}{r}\binom{r}{s} = \binom{n}{s}\binom{n-s}{r-s}$. Now the entropy of A is at most $\binom{n}{s}$, and so our lower bound on the remaining entropy is $(\log \binom{r}{s} - 2t \log n)$. To make this quantity large requires making r very large.

MODIFIED JS FINGERPRINTS. We suggest a modification of the Juels-Sudan scheme with entropy loss at most $2t \log n$ and storage $s \log n$. Our scheme has the advantage of being even simpler to analyze. As before, we assume n is a prime power and work over $\mathcal{F} = GF(n)$. An intuition for the scheme is that the numbers $y_{s+1}, ..., y_r$ from the JS scheme need not be chosen at random. One can instead evaluate them as $y_i = p'(x_i)$ for some polynomial p'. One can then represent the entire list of pairs (x_i, y_i) using only the coefficients of p'.

Algorithm 3 (Modified JS Fuzzy Fingerprint). Input: a set $A \subseteq \mathcal{U}$.

1. Choose $p()$ at random from the set of polynomials of degree at most $k = s - 2t - 1$ over $\mathcal{F}$.
2. Let $p'()$ be the unique monic polynomial of degree exactly s such that $p'(x) = p(x)$ for all $x \in A$. (Write $p'(x) = x^s + \sum_{i=0}^{s-1} a_i x^i$. Solve for $a_0, ..., a_{s-1}$ using the s linear constraints $p'(x) = p(x)$, $x \in A$.)

3. Output the list of coefficients of $p'()$, that is $\mathsf{FF}(A) = (a_0, ..., a_{s-1})$.

First, observe that solving for $p'()$ in Step 2 is always possible, since the s constraints $\sum_{i=0}^{s-1} a_i x^i = p(x) - x^s$ are in fact linearly independent (this is just polynomial interpolation).

Second, this fingerprint scheme can tolerate t set difference errors. Suppose we are given a set $B \subseteq \mathcal{U}$ which agrees with A in at least $s - t$ positions. Given $p' = \mathsf{FF}(A)$, one can evaluate p' on all the points in the set B. The resulting vector agrees with p on at least $s - t$ positions, and using the decoding algorithm for Reed-Solomon codes, one can thus reconstruct p exactly (since $k = s - 2t - 1$). Finally, the set A can be recovered by finding the roots of the polynomial $p' - p$: since $p' - p$ is not identically zero and has degree exactly s, it can have at most s roots and so $p' - p$ is zero only on A.

We now turn to the entropy loss of the scheme. The fingerprinting scheme invests $(s-2t)\log n$ bits of randomness to choose the polynomial p. The number of possible outputs p' is n^s. If X is the invested randomness, then the (average) min-entropy (A, X) given $\mathsf{FF}(A)$ is at least $\tilde{\mathbf{H}}_\infty(A) - 2t \log n$. The randomness X can be recovered from A and $\mathsf{FF}(A)$, and so we have $\tilde{\mathbf{H}}_\infty(A \mid \mathsf{FF}(A)) \geq \tilde{\mathbf{H}}_\infty(A) - 2t \log n$. We have proved:

Lemma 5.3 (Analysis of Modified JS). *The entropy loss of the modified JS scheme is at most $2t \log n$. The scheme has storage $(s+1)\log n$ for sets of size s in $[n]$, and both the fingerprint generation $\mathsf{FF}()$ and the recovery procedure $\mathsf{Rec}()$ run in polynomial time.*

The short length of the fingerprint makes this scheme feasible for essentially any ratio of set size to universe size (we only need $\log n$ to be polynomial in s). Moreover, for large universes the entropy loss $2t \log n$ is essentially optimal for the uniform case $m = \log \binom{n}{s}$. Lemma A.1 shows that for a uniformly distributed input, the best possible entropy loss is $m - m' \geq \log \binom{n}{s} - \log A(n, s, 4t+1)$, where $A(n, s, d)$ is the maximum size of a code of constant weight s and minimum Hamming distance d. Using a bound of Agrell *et al* ([1], Theorem 12), the entropy loss is at least:

$$m - m' \geq \log \binom{n}{s} - \log A(n, s, 4t+1) \geq \log \binom{n}{s} - \log \left(\binom{n}{s-2t} \Big/ \binom{s}{s-2t} \right) = \log \binom{n-s+2t}{2t}$$

When $n \geq s$, this last quantity is roughly $2t \log n$, as desired.

5.3 Set Difference via the Hamming Metric: Sublinear-Time Decoding

In this section, we show that code-offset construction can in fact be adapted for small sets in large universe, using specific properties of algebraic codes. We will show that BCH codes, which contain Hamming and Reed-Solomon codes as special cases, have these properties.

SYNDROMES OF LINEAR CODES. For a $[n, k, d]$ linear code C with parity check matrix H, recall that the syndrome of a word $w \in \{0,1\}^n$ is $\mathsf{syn}(w) = Hw$. The syndrome has length $n - k$, and the code is exactly the set of words c such that $\mathsf{syn}(c) = 0^{n-k}$. The syndrome captures all the information necessary for decoding. That is, suppose a codeword c is sent through a channel and the word $w = c \oplus e$ is received. First, the syndrome of w is the syndrome of e: $\mathsf{syn}(w) = \mathsf{syn}(c) \oplus \mathsf{syn}(e) = 0 \oplus \mathsf{syn}(e) = \mathsf{syn}(e)$. Moreover, for any value u, there is at most one word e of weight less than $d/2$ such that $\mathsf{syn}(e) = u$ (the existence of a pair of distinct words e_1, e_2 would mean that $e_1 + e_2$ is a codeword of weight less than d). Thus, knowing syndrome $\mathsf{syn}(w)$ is enough to determine the error pattern e if not too many errors occurred.

As mentioned before, we can reformulate the code-offset construction in terms of syndrome: $\mathsf{FF}(w) = \mathsf{syn}(w)$. The two schemes are equivalent: given $\mathsf{syn}(w)$ one can sample from $w \oplus C(X)$ by choosing a random string v with $\mathsf{syn}(v) = \mathsf{syn}(w)$; conversely, $\mathsf{syn}(w \oplus C(X)) = \mathsf{syn}(w)$. This reformulation gives us no special advantage when the universe is small: storing $w + C(X)$ is not a problem. However, it's a substantial improvement when $n \gg n - k$.

SYNDROME MANIPULATION FOR SMALL-WEIGHT WORDS. Suppose now that we have a small set $A \subseteq [n]$ of size s, where $n \gg s$. Let $x_A \in \{0,1\}^n$ denote the characteristic vector of A. If we want to use $\mathsf{syn}(x_A)$ as the fingerprint of A, then we must choose a code with $n - k \leq \log \binom{n}{s} \approx s \log n$, since the fingerprint has entropy loss $(n-k)$ and the maximum entropy of A is $\log \binom{n}{s}$.

Binary BCH codes are a family of $[n, k, d]$ linear codes with $d = 4t + 1$ and $k = n - 2t \log n$ (assuming $n + 1$ is a power of 2) (see, e.g. [18]). These codes are optimal for $t \ll n$ by the Hamming bound, which implies that $k \leq n - \log \binom{n}{2t}$ [18]. Using the code-offset fingerprint with a BCH code C, we get entropy loss $n - k = 2t \log n$, just as we did for the modified Juels-Sudan scheme (recall that $d \geq 4t+1$ allows us to correct t set difference errors).

The only problem is that the scheme appears to require computation time $\Omega(n)$, since we must compute $\mathsf{syn}(x_A) = Hx_A$ and, later, run a decoding algorithm to recover x_A. For BCH codes (including Hamming codes), this difficulty can be overcome. A word of small weight x can be described by listing the positions on which it is nonzero. We call this description the *support* of x and write $\mathsf{supp}(x)$ (that is $\mathsf{supp}(x_A) = A$)).

Lemma 5.4. *For a $[n, k, d]$ binary BCH code C one can compute:*

1. $\mathsf{syn}(x)$*, given* $\mathsf{supp}(x)$*, and*
2. $\mathsf{supp}(x)$*, given* $\mathsf{syn}(x)$ *(when x has weight at most $(d-1)/2$),*

in time polynomial in $|\mathsf{supp}(x)| = weight(x) \cdot \log(n)$ and $|\mathsf{syn}(x)| = n - k$.

The proof of Lemma 5.4 mainly requires a careful reworking of the standard BCH decoding algorithm. The details of BCH codes and Lemma 5.4 are presented in Appendix B. For now, we present the resulting fingerprinting scheme for set difference. The algorithm works in the field $GF(2^m) = GF(n+1)$, and assumes a generator α for $GF(2^m)$ has been chosen ahead of time.

Algorithm 4 (BCH-based Fuzzy Fingerprint). Input: a set $A \in [n]$ of size s, where $n = 2^m - 1$.
(Here α is a generator for the field $GF(2^m)$ which is fixed ahead of time.)

1. Let $p(x) = \sum_{i \in A} x^i$.
2. Output $\mathsf{FF}(A) = (p(\alpha), p(\alpha^3), p(\alpha^5), ..., p(\alpha^{4t+1}))$ (computations done in $GF(2^m)$).

The algorithm $\mathsf{Rec}()$ which recovers A from $\mathsf{FF}(A)$ and any set which intersects A in at least $s - t$ points is explained in Appendix B. However, the bound on entropy loss is easy to see: the output is $2t \log n$ bits long, and hence the entropy loss is at most $2t \log n$. We obtain:

Theorem 5.5. *The BCH scheme above is a $[m, m - 2t \log n, t]$ fuzzy fingerprint scheme for set difference with storage $2t \log n$. The algorithms* FF *and* Rec *both run in polynomial time.*

6 Constructions for Edit Distance

First we note that simply applying the same approach as we took for the transitive metric spaces before (the Hamming space and the set difference space for small universe sizes) does not work here, because the edit metric does not seem to be transitive. Indeed, it is unclear how to build a permutation π such that for any w' close to w, we also have $\pi(w')$ close to $x = \pi(w)$. For example, setting $\pi(y) = y \oplus (x \oplus w)$ is easily seen not to work with insertions and deletions. Similarly, if I is some sequence of insertions and deletions mapping w to x, it is not true that applying I to w' (which is close to w) will necessarily result in some x' close to x. In fact, then we could even get $\mathsf{dis}(w', x') = 2\mathsf{dis}(w, x) + \mathsf{dis}(w, w')$.

Perhaps one could try to simply embed the edit metric into the Hamming metric using known embeddings, such as conventionally used low-distorion embdeddings, which provide that all distances are preserved up to some small

"distortion" factor. However, there are no known nontrivial low-distortion embeddings from the edit metric to the Hamming metric. Moreover, it was recently proved by Andoni et al [2] that no such embedding can have distortion less than $3/2$, and it was conjectured that a much stronger lower bound should hold.

Thus, as the previous approaches don't work, we turn to the embdeddings we defined specifically for fuzzy extractors: biometric embeddings. Unlike low-distortion embeddings, biometric embeddings do not care about relative distances, as long as points that were "close" (closer than t_1) do not become "distant" (farther apart than t_2). The only additional requirement of biometric embeddings is that they preserve some min-entropy: we do not want too many points to collide together, although collisions are allowed, even collisions of distant points. We will build a biometric embedding from the edit distance to the set difference.

A *c-shingle* [5], which is a length-c consecutive substring of a given string w. A *c-shingling* [5] of a string w of length n is the set (ignoring order or repetition) of all $(n-c+1)$ c-shingles of w. Thus, the range of the c-shingling operation consists of all nonempty subsets of size at most $n-c+1$ of $\{0,1\}^c$. To simplify our future computations, we will always arbitrarily pad the c-shingling of any string w to contain precisely n distinct shingles (say, by adding the first $n - |c\text{-shingling}|$ elements of $\{0,1\}^c$ not present in the given c-shingling). Thus, we can define a deterministic map $\mathsf{SH}_c(w)$ which maps w into n substrings of $\{0,1\}^c$, where we assume that $c \geq \log_2 n$. Let $\mathsf{Edit}(n)$ stand for the edit metric over $\{0,1\}^n$, and $\mathsf{SDif}(N,s)$ stand for the set difference metric over $[N]$ where the set sizes are s. We now show that c-shingling yields pretty good biometric embeddings for our purposes.

Lemma 6.1. *For any $c > \log_2 n$, SH_c is a $(t_1, t_2 = ct_1, m_1, m_2 = m_1 - \frac{n \log_2 n}{c})$-biometric embedding of $\mathsf{Edit}(n)$ into $\mathsf{SDif}(2^c, n)$.*

Proof. Assume $\mathsf{dis}(w_1, w_1')_{ed} \leq t_1$ and that I is the smallest set of $2t_1$ insertions and deletions which transforms w into w'. It is easy to see that each character deletion or insertion affects at most c shingles, and thus the symmetric difference between $\mathsf{SH}_c(w_1)$ and $\mathsf{SH}_c(w_2) \leq 2ct_1$, which implies that $\mathsf{dis}(\mathsf{SH}_c(w_1), \mathsf{SH}_c(w_2))_{sd} \leq ct_1$, as needed.

Now, assume w_1 is any string. Define $g_c(w_1)$ as follows. One computes $\mathsf{SH}_c(w_1)$, and stores n resulting shingles in lexicographic order $h_1 \ldots h_n$. Next, one naturally partitions w_1 into n/c disjoint shingles of length c, call them $k_1 \ldots k_{n/c}$. Next, for $1 \leq j \leq n/c$, one sets $p_c(j)$ to be the index $i \in \{1 \ldots n\}$ such that $k_j = h_i$. Namely, it tells the index of the j-th disjoint shingle of w_1 in the ordered n-set $\mathsf{SH}_c(w_1)$. Finally, one sets $g_c(w_1) = (p_c(1) \ldots p_c(n/c))$. Notice, the length of $g_c(w_1)$ is $\frac{n}{c} \cdot \log_2 n$, and also that w_1 can be completely recovered from $\mathsf{SH}_c(w_1)$ and $g_c(w_1)$.

Now, assume W_1 is any distribution of min-entropy at least m_1 on $\mathsf{Edit}(n)$. Since $g_c(W)$ has length $(n \log_2 n/c)$, its min-entropy is at most this much as well. But since min-entropy of W_1 drops to 0 when given $\mathsf{SH}_c(W_1)$ and $g_c(W_1)$, it means that the min-entropy of $\mathsf{SH}_c(W_1)$ must be at least $m_2 \geq m_1 - (n \log_2 n)/c$, as claimed. □

We can now optimize the value c. By either Lemma 5.3 or Theorem 5.5, for arbitrary universe size (in our case 2^c) and distance threshold $t_2 = ct_1$, we can construct a fuzzy fingerprint for the set difference metric with min-entropy loss $2t_2 \log_2(2^c) = 2t_1c^2$, which leaves us total min-entropy $m_2' = m_2 - 2t_1c^2 \geq m_1 - \frac{n \log n}{c} - 2t_1c^2$. Applying further Lemma 3.1, we can convert it into a fuzzy extractor over $\mathsf{SDif}(2^c, n)$ for the min-entropy level m_2 with error ϵ, which can extract at least $\ell = m_2' - 2\log\left(\frac{1}{\epsilon}\right) \geq m_1 - \frac{n \log n}{c} - 2t_1c^2 - 2\log\left(\frac{1}{\epsilon}\right)$ bits, while still correcting $t_2 = ct_1$ of errors in $\mathsf{SDif}(2^c, n)$. We can now apply Lemma 3.3 to get an $(\mathsf{Edit}(n), m_1, m_1 - \frac{n \log n}{c} - 2t_1c^2 - 2\log\left(\frac{1}{\epsilon}\right), t_1, \epsilon)$-fuzzy extractor. Let us now optimize for the value of $c \geq \log_2 n$. We can set $\frac{n \log n}{c} = 2t_1c^2$, which gives $c = (\frac{n \log n}{2t_1})^{1/3}$. We get $\ell = m_1 - (2t_1 n^2 \log^2 n)^{1/3} - 2\log\left(\frac{1}{\epsilon}\right)$ and therefore

Theorem 6.2. *There exists an efficient $(\mathsf{Edit}(n), m_1, m_1 - (2t_1 n^2 \log^2 n)^{1/3} - 2\log\left(\frac{1}{\epsilon}\right), t_1, \epsilon)$ fuzzy extractor. Setting $t_1 = m_1^3/(16n^2 \log^2 n)$, we get an efficient $(\mathsf{Edit}(n), m_1, \frac{m_1}{2} - 2\log\left(\frac{1}{\epsilon}\right), \frac{m_1^3}{16n^2 \log^2 n}, \epsilon)$ fuzzy extractor. In particular, if $m_1 = \Omega(n)$, one can extract $\Omega(n)$ bits while tolerating $\Omega(n/\log^2 n)$ insertions and deletions.*

Acknowledgments

We thank Piotr Indyk for discussions about embeddings and for his help in the proof of Lemma 6.1. We are also thankful to Madhu Sudan for helpful discussions about the construction of [16] and the uses of error-correcting codes. Finally, we thank Rafi Ostrovsky for discussions in the initial phases of this work.

The work of the first author was partly funded by the National Science Foundation under CAREER Award No. CCR-0133806 and Trusted Computing Grant No. CCR-0311095, and by the New York University Research Challenge Fund 25-74100-N5237. The work of the second author was partly funded by the National Science Foundation under Grant No. CCR-0311485. The work of the third author was partly funded by US A.R.O. grant DAAD19-00-1-0177 and by a Microsoft Fellowship.

References

[1] E. Agrell, A. Vardy, and K. Zeger. Upper bounds for constant-weight codes. *IEEE Transactions on Information Theory*, vol. 46, no. 7, pp. 2373–2395, 2000.

[2] A. Andoni, M. Deza, A. Gupta, P. Indyk, S. Raskhodnikova. Lower bounds for embedding edit distance into normed spaces. In *Proc. of SODA*, pp. 523–526, 2003.

[3] C. Bennett, G. Brassard, and J. Robert. Privacy Amplification by Public Discussion. In *SIAM J. on Computing*, 17(2), pp. 210–229, 1988.

[4] C. Bennett, G. Brassard, C. Crépeau, and U. Maurer. Generalized Privacy Amplification. In *IEEE Trans. Info. Theory*, 41(6), pp. 1915-1923, 1995.

[5] A. Broder. On the resemblence and containment of documents. In *Compression and Complexity of Sequences* (SEQUENCES), 1997.

[6] A. E. Brouwer, J. B. Shearer, N. J. A. Sloane, and W. D. Smith, "A new table of constant weight codes," IEEE Trans. Inform. Theory, vol. 36, pp. 1334–1380, Nov. 1990.

[7] C. Crépeau. Efficient Cryptographic Protocols Based on Noisy Channels. In *Eurocrypt 1997*, pp. 306–317, 1997.

[8] G. Davida, Y. Frankel, B. Matt. On enabling secure applications through off-line biometric identification. In *Proc. Symp. on Security and Privacy*, pp. 148–157, 1998.

[9] C. Ellison, C. Hall, R. Milbert, B. Schneier. Protecting Keys with Personal Entropy. *Future Generation Computer Systems*, 16:311–318, 2000.

[10] N. Frykholm. Passwords: Beyond the Terminal Interaction Model. *Master's Thesis*, Umea University. Available at `http://www.cs.umu.se/~niklasf/exjobb/`.

[11] N. Frykholm, A. Juels. Error-Tolerant Password Recovery. In *Proc. of ACM Conference on Computer and Communications Security*, pp. 1–8, 2001.

[12] V. Guruswami, M. Sudan. Improved Decoding of Reed-Solomon and Algebraic-Geometric Codes. In *Proc. of FOCS*, pp. 28–39, 1998.

[13] J. Håstad, R. Impagliazzo, L. Levin, M. Luby. A Pseudorandom generator from any one-way function. In *Proc. of STOC*, 1989.

[14] A. Juels, B. Mannal. Survey of Biometric Authentication Technologies. Available at `http://www.rsalabs.com/staff/ajuels/`.

[15] A. Juels, M. Wattenberg. A Fuzzy Commitment Scheme. In *Proc. of ACM Conference on Computer and Communications Security*, pp. 28–36, 1999.

[16] A. Juels and M. Sudan. A Fuzzy Vault Scheme. *IEEE International Symposium on Information Theory*, 2002.

[17] J. Kelsey, B. Schneier, C. Hall, D. Wagner. Secure Applications of Low-Entropy Keys. In *Proc. of Information Security Workshop*, pp. 121–134, 1997.

[18] J.H. van Lint. *Introduction to Coding Theory.* Springer-Verlag, 1992, 183 pp.

[19] F. Monrose, M. Reiter, S. Wetzel. Password Hardening Based on Keystroke Dynamics. In *Proc. ACM Conference on Computer and Communications Security*, pp. 73–82, 1999.

[20] F. Monrose, M. Reiter, Q. Li, S. Wetzel. Cryptographic key generation from voice. In *Proc. IEEE Symposium on Security and Privacy*, 2001.

[21] F. Monrose, M. Reiter, Q. Li, S. Wetzel. Using voice to generate cryptographic keys. In *Proc. of Odyssey 2001, The Speaker Verification Workshop*, 2001.

[22] R. Morris, K. Thomson. Password Security: a case history. In *Communications of the ACM*, 22(11):594–597, 1979.

[23] N. Nisan, A. Ta-Shma. Extracting Randomness: a survey and new constructions. In *JCSS*, 58(1):148–173, 1999.

[24] N. Nisan, D. Zuckerman. Randomness is Linear in Space. In *JCSS*, 52(1):43–52, 1996.

[25] J. Radhakrishnan and A. Ta-Shma. Tight bounds for depth-two superconcentrators. In Proc. of FOCS, pp. 585–594, 1997.

[26] R. Shaltiel. Recent developments in Explicit Constructions of Extractors. *Bulletin of the EATCS*, 77:67–95, 2002.

[27] A. Shamir. How to share a secret. In *Communic. of the ACM*, 22:612-613, 1979.

[28] V. Shoup. A Proposal for an ISO Standard for Public Key Encryption. Available at `http://eprint.iacr.org/2001/112`, 2001.

A Lower Bounds from Coding

Recall that an $(\mathcal{M}, K, t)$ code is a subset of the metric space $\mathcal{M}$ which can *correct* t errors (this is slightly different from the usual notation of the coding theory literature.

Let $K(\mathcal{M}, t)$ be the largest K for which there exists and $(\mathcal{M}, K, t)$-code. Given any set S of 2^m points in $\mathcal{M}$, we let $K(\mathcal{M}, t, S)$ be the largest K such that there exists an $(\mathcal{M}, K, t)$-code all of whose K points belong to S. Finally, we let $L(\mathcal{M}, t, m) = \log(\min_{|S|=2^m} K(n, t, S))$. Of course, when $m = \log|\mathcal{M}| = \log N$, we get $L(\mathcal{M}, t, n) = \log K(\mathcal{M}, t)$. The exact determination of quantities $K(\mathcal{M}, t)$ and $K(\mathcal{M}, t, S)$ form the main problem of coding theory, and is typically very hard. To the best of our knowledge, the quantity $L(\mathcal{M}, t, m)$ was not explicitly studied in any of three metrics that we study, and its exact determination seems very hard as well.

We give two simple lower bounds (one for fuzzy fingerprints, the other for fuzzy extractors) which, somewhat surprisingly, show that our constructions for the Hamming and Set Difference metrics are essentially optimal, at least when the original input distribution is uniform.

Lemma A.1. *The existence of $(\mathcal{M}, m, m', t)$ fuzzy fingerprint implies that $m' \leq L(\mathcal{M}, t, m)$. In particular, when $m = \log N$ (i.e., when the password is truly uniform), $m' \leq \log K(\mathcal{M}, t)$.*

Proof. Assume FF is such fuzzy fingerprint. Let S be any set of size 2^m in $\mathcal{M}$, and let W be uniform over S. Then we must have $\tilde{\mathbf{H}}_\infty(W \mid \mathsf{FF}(W)) \geq m'$. In particular, there must be some particular value v such that $\mathbf{H}_\infty(W \mid \mathsf{FF}(W) = v) \geq m'$. But this means that conditioned on $\mathsf{FF}(W) = v$, there are at least $2^{m'}$ points w in S (call this set T) which could produce $\mathsf{FF}(W) = v$. We claim that these $2^{m'}$ values of w form a code of error-correcting distance t. Indeed, otherwise there would be a point $w' \in \mathcal{M}$ such that $\mathsf{dis}(w_0, w') \leq t$ and $\mathsf{dis}(w_1, w') \leq t$ for some $w_0, w_1 \in T$. But then we must have that $\mathsf{Rec}(w', v)$ is equal to both w_0 and w_1, which is impossible. Thus, the set T above must form an $(\mathcal{M}, 2^{m'}, t)$-code inside S, which means that $m' \leq \log K(\mathcal{M}, t, S)$. Since S was arbitrary, the bound follows. □

Lemma A.2. *The existence of $(\mathcal{M}, m, \ell, t, \epsilon)$-fuzzy extractors implies that $\ell \leq L(\mathcal{M}, t, m) - \log(1 - \epsilon)$. In particular, when $m = \log N$ (i.e., when the password is truly uniform), $\ell \leq \log K(\mathcal{M}, t) - \log(1 - \epsilon)$.*

Proof. Assume $(\mathsf{Gen}, \mathsf{Rep})$ is such a fuzzy extractor. Let S be any set of size 2^m in $\mathcal{M}$, and let W be uniform over S. Then we must have $\mathbf{SD}\left(\langle R, P\rangle, \langle U_\ell, P\rangle\right) \leq \epsilon$. In particular, there must be some particular value p of P such that R is ϵ-close to U_ℓ conditioned on $P = p$. In particular, this means that conditioned on $P = p$, there are at least $(1 - \epsilon)2^\ell$ points $r \in \{0, 1\}^\ell$ (call this set T) which could be extracted with $P = p$. Now, map every $r \in T$ to some arbitrary $w \in S$ which could have produced r with nonzero probability given $P = p$, and call this map C. We claim that C must define a code with error-correcting distance t. Indeed, otherwise there would be a point $w' \in \mathcal{M}$ such that $\mathsf{dis}(C(r_1), w') \leq t$ and $\mathsf{dis}(C(r_2), w') \leq t$ for some $r_1 \neq r_2$. But then we must have that $\mathsf{Rep}(w', p)$ is equal to both r_1 and r_2, which is impossible. Thus, the map C above must form an $(M, 2^{\ell + \log(1-\epsilon)}, t)$-code inside S, which means that $\ell \leq \log K(\mathcal{M}, t, S) - \log(1 - \epsilon)$. Since S was arbitrary, the bound follows. □

Observe that, as long as $\epsilon < 1/2$, we have $0 < -\log(1 - \epsilon) < 1$, so the lowerbounds on fuzzy fingerprints and fuzzy extractors differ by less than a bit.

B Syndrome Decoding in Sublinear Time

We show that the standard decoding algorithm for BCH codes can be modified to run in time polynomial in the length syndrome. This works for BCH codes over any field $GF(q)$, which include Hamming codes in the binary case and Reed-Solomon for the case $q > n$. BCH codes are handled in detail in many textbooks (e.g., [18]); our presentation here is quite terse. For simplicity, we only discuss primitive, narrow-sense BCH codes here; the discussion extends easily to the general case.

Definition 5. *Let $n = q^m - 1$. The narrow-sense, primitive BCH code of designed distance δ and length n over $GF(q)$ is given by the set of vectors $(c_0, ..., c_{n-1}) \in GF(q)^n$ such that $0 = c(\alpha^1) = c(\alpha^2) = \cdots = c(\alpha^{\delta-1})$, where $c(x) = \sum_{i=0}^{n-1} c_i x^i$ is a polynomial over $GF(q^m)$ and α is a fixed generator for $GF(q^m)$.*

Readers may be used to seeing a different definition of these codes: one can also take the set of vectors $(p(\alpha), \ldots, p(\alpha^{n-1}))$ given by polynomials with coefficients in $GF(q^m)$ all of whose values lie in $GF(q) \subseteq GF(q^m)$. The equivalence of the various definitions is standard.

Returning to Definition 5, we can see that some coniditions are redundant. Specifically, because $c_i \in GF(q)$, we have $c(x^q) = (c(x))^q$. Thus, we discard a $1/q$ fraction of the conditions on c without changing the code. The

syndrome of a word w can thus be computed as the vector of values $w(\alpha^i)$, where $1 \leq i < \delta$ and $q \not| i$ and the polynomial $w()$ is given by $w(x) = \sum_{i=0}^{n-1} w_i x^i$. Each constraint $c(\alpha^i) = 0$ on codewords results in at most m constraints (in the field $GF(q)$) over the vector c.

Fact B.1. *The q-ary BCH code of length n and designed distance d has dimension $k \geq n - m(\delta-1) + (1/q)m \lfloor \delta/q \rfloor$ and minimum distance $d \geq \delta$.*

The parity check matrix for a BCH code is given by:

$$H = \begin{pmatrix} \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ & \vdots & & \\ \alpha^{\delta-1} & \alpha^{2(\delta-1)} & \cdots & \alpha^{(n-1)(\delta-1)} \end{pmatrix}$$

As mentioned above, all rows for α^i with $q|i$ are redundant. Nevertheless, the first part of Lemma 5.4 is clear: to compute the syndrome of a low-weight word, one need only be able to compute any given column of the parity check matrix in time polynomial in the length of the column. In fact, the computation time is almost linear, since one only needs to do one exponentiation and δ multiplications to compute any particular row.

A low-weight word $p \in GF(q)^n$ can be represented either as a long string or, more compactly, as a list of positions where it is nonzero and its values at those points. We call this representation the support list of x and denote it $\mathsf{supp}(p) = \{(i, p_i)\}_{i:p_i \neq 0}$.

Lemma B.2. *For a q-ary BCH code C of designed distance δ, one can compute:*

1. $\mathsf{syn}(p)$*, given* $\mathsf{supp}(p)$*, and*
2. $\mathsf{supp}(p)$*, given* $\mathsf{syn}(p)$ *(when p has weight at most $(\delta-1)/2$),*

in time polynomial in $|\mathsf{supp}(p)| = weight(p) \cdot \log(n) \cdot \log(q)$ *and* $|\mathsf{syn}(p)| = (n-k)\log q$.

Proof. Recall that $\mathsf{syn}(p) = (p(\alpha), ..., p(\alpha^{\delta-1}))$ where $p(x) = \sum_{i=0}^{n-1} p_i$. Part (1) is easy, since any column of the parity check matrix H can be computed with one exponentiation and $\delta - 1$ multiplications in $GF(q^m)$. For (2), we carefully analyze the various steps of the standard BCH decoding algorithm, based on its presentation in [18]. Let $M = \{i | p_i \neq 0\}$, and define

$$\sigma(z) = \prod_{i \in M} (1 - \alpha^i z) \quad \text{and} \quad \omega(z) = \sigma(z) \sum_{j \in M} \frac{p_j \alpha^j}{(1 - \alpha^j z)}$$

Since $(1 - \alpha^j z)$ divides $\sigma(z)$ for $j \in M$, we see that $\omega(z)$ is in fact a polynomial of degree at most $|M| = weight(p) \leq (\delta - 1)/2$. The polynomials $\sigma(z)$ and $\omega(z)$ are known as the error locator polynomial and evaluator polynomial, respectively. If we now consider formal power series over $GF(q^m)$, we get:

$$\frac{\omega(z)}{\sigma(z)} = \sum_{j \in B} \frac{p_j \alpha^j z}{(1 - \alpha^j z)} = \sum_{\ell=1}^{\infty} z^\ell p(\alpha^\ell)$$

We are given $S_\ell = p(\alpha_\ell)$ for $\ell = 1, ..., \delta$. Let $S(z) = \sum_{\ell=1}^{\delta-1} S_\ell z^\ell$. The equation above implies that $S(z)\sigma(z) \equiv \omega(z) \mod z^\delta$. The solution $\omega(), \sigma()$ is "unique" in the following sense: any other solution $\omega'(z), \sigma'(z)$ satisfies $w'(z)/\sigma'(z) = \omega(z)/\sigma(z)$. Multiplying the initial congruence by $\sigma'()$ yields $\omega(z)\sigma'(z) \equiv \sigma(z)\omega'(z) \mod z^\delta$. Since the both sides of the congruence have degree at most $\delta - 1$, they are in fact equal as polynomials.

Thus it is sufficient to find any solution $\sigma'(), \omega'()$ to the congruence $S(z)\sigma'(z) = \omega'(z) \mod z^{\delta}$ and reduce the resulting fraction $\omega'(z)/\sigma'(z)$ to botain a solution $\omega(), \sigma()$ of minimal degree. Finally, the roots of $\sigma(z)$ are the points α^{-i} for $i \in M$, and the exact value of p_i can be recovered using the equation $\omega(\alpha^{-i}) = p_i \prod_{j \in M, j \neq i}(1 - \alpha^{j-i})$.

Solving the congruence only requires solving a system of $\delta - 1$ linear equations, which is certainly polynomial in $\delta \log q$. The reduction of the fraction $\omega'(z)/\sigma'(z)$ requires only running Euclid's algorithm for finding the g.c.d. of two polynomials. Finally, finding the roots of $\sigma()$ can be done in time polynomial in the degree of $\sigma()$, which is at most $\delta/2$. □