Towards Practical Biometric Key Generation with Randomized Biometric Templates

Lucas Ballard*
Google, Inc.
lucasballard@google.com

Seny Kamara*
Microsoft Research
senyk@microsoft.com

Fabian Monrose*
UNC Chapel Hill
fabian@cs.unc.edu

Michael K. Reiter
UNC Chapel Hill
reiter@cs.unc.edu

ABSTRACT

Although biometrics have garnered significant interest as a source of entropy for cryptographic key generation, recent studies indicate that many biometric modalities may not actually offer enough uncertainty for this purpose. In this paper, we exploit a novel source of entropy that can be used with any biometric modality but that has yet to be utilized for key generation, namely associating uncertainty with the way in which the biometric input is measured. Our construction poses only a modest requirement on a user: the ability to remember a low-entropy password. We identify the technical challenges of this approach, and develop novel techniques to overcome these difficulties. Our analysis of this approach indicates that it may offer the potential to generate stronger keys: In our experiments, 40% of the users are able to generate keys that are at least 2^{30} times stronger than passwords alone.

Categories and Subject Descriptors

E.3 [**Data Encryption**]; H.1 [**Models and Principles**]: User/Machine Systems

General Terms

Security, Design

Keywords

Biometrics, Cryptographic Keys

1. INTRODUCTION

Humans are unable to generate and remember strong secrets, and thus have difficulty managing cryptographic keys [1, 10]. To address this problem, numerous proposals have been suggested to enable people to reliably generate high-entropy cryptographic keys from their *biometrics*, or, measurements of their physiology or behavior. These *Biometric Cryptographic Key Generators* (BKGs) are believed to be useful as they allow users to seamlessly recreate strong keys. Unfortunately, despite interest in BKGs (e.g. [16, 15, 8, 7]), recent studies (e.g., [4, 19, 21, 3]) have shown that some biometric modalities may be too weak to offer enough security for key generation. To combat this problem, we explore new techniques of extracting entropy from biometrics

In this paper we present a novel way to think about biometrics and propose a new BKG that exploits a source of randomness that, to our knowledge, has not been previously used to strengthen keys. We suggest adding uncertainty to the way that a BKG *measures* the biometric for each user. To reproduce the correct key, an adversary must guess both the biometric input and the statistical *features* that were used to measure the user. This approach both increases the entropy of the keys and reduces the susceptibility of the BKG to forgery. By carefully selecting *strong* features (i.e., those that are easier for a specific user to replicate) we are able to reduce the error-tolerance of each feature, and thus increase resistance to forgery.

To achieve our goals we propose *Randomized Biometric Templates* (RBTs), templates that can be used by legitimate users to create keys, but are designed so that attackers cannot learn how to measure biometric inputs. RBTs assign different features to different users, and encode the features so that adversaries cannot determine which features were originally used to generate a key. The utility of this approach is two-fold. First, it increases the work required to search for the correct key because an attacker must guess both the set of features that were used, as well as the correct biometric sample. Second, we are able to assign only strong features to each user, so an attacker must provide a more precise guess of the biometric input to correctly recreate the key.

In this paper we describe how to construct RBTs for any biometric modality. We describe both the cryptographic construction (Section 5) and the statistical process of selecting features (Section 6). As we show, feature selection is non-trivial, but of the utmost importance. We are able to craft algorithms that assign only high-quality features to each user, but in a way that appears random to an adversary. We provide arguments that RBTs are secure (Section 7). Additionally, we empirically evaluate RBTs with recently-proposed standards (Section 8). In particular, our empirical evaluation focuses on an (arguably) weak biometric modality, and we are able to show that for many users, our techniques are able to extract more entropy than existing approaches. This provides evidence that extracting entropy from the feature selection process can improve the security afforded by BKGs.

*Research conducted at Johns Hopkins University, Baltimore, MD.

CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.

2. RELATED WORK

Cryptographic keys can be derived from any number of biometric *modalities*. For instance, one can collect biometric readings of an iris [11], fingerprints [18, 20], or the geometry of a face. These are examples of *physiological* biometrics as they measure concrete aspects of a person's biological traits. *Behavioral* biometrics, on the other hand, measure how people perform actions. For example, one could exploit properties of how a phrase is spoken [15] or written [22], and even how people think. Behavioral biometrics are attractive for key generation because while physiological biometrics cannot change, behavioral biometrics change with the action that is performed. This allows users to create different keys.

Monrose et al. proposed the first practical system that uses behavioral (versus physiological) biometrics in key generation [16]. Their technique uses keystroke latencies to increase the entropy of standard passwords. They show that their system yields keys that are at least as strong as the password alone, and in some instances their approach increases the workload of an attacker by a multiplicative factor of 2^{15}. The construction makes the important distinction of identifying which features, in this case, which keystroke latencies, are useful (i.e., "distinguishing") for a specific user, and which are not. To do so, it sets a global threshold for each feature, and extracts a bit for each user by assigning a 0 or 1 if her measurements are consistently below or above that threshold. If the user does not provide measurements that fall consistently on one side of the threshold, then the system ignores that feature for that user.

RBTs are outwardly similar to the constructions of Monrose et al., although they offer several advantages. First, although our approach also uses quantization for error correction, we partition the range of each feature into more than two segments, and hence we can achieve higher entropy rates and lower False Accept Rates. Second, our notion of a "distinguishing" feature (i.e., the features that we choose to assign to each user) is more flexible. We assign a feature to a user if she can reliably repeat that feature, even if the mean value falls anywhere in the feature's range. By contrast, the construction of Monrose et al. ignores features that can be repeated reliably, but whose mean falls directly on the global threshold. Finally, our approach is based on block ciphers and hash functions, which are more computationally efficient than the number theoretic primitives that were used in that work.

There has also been work in "Fuzzy Cryptography." The idea of Fuzzy Cryptography was first introduced by Juels and Wattenberg [13], who describe a commitment scheme that supports noise-tolerant decommitments. Further work included a Fuzzy Vault [12], which was later classified as an instance of a Secure Sketch, which can be used to build a Fuzzy Extractor [8, 6]. Fuzzy Extractors treat biometrics as high-entropy, error-prone sources and apply error-correction algorithms and randomness extractors to generate strings that are close to random. As opposed to fuzzy extractors, which distill the entropy of a biometric, our approach can be seen as distilling the entropy of the biometric *and* the features used to measure it.

Recently we proposed techniques to evaluate BKGs. We examined the common practice of using weak forgeries to analyze BKGs [4] and advocated using more realistic "trained" forgers to provide a better estimate of security. We also explored "generative algorithms" as a way of using partial knowledge about a target user's biometric to create forgeries. Finally, we described a set of necessary security requirements and proposed an algorithm that probabilistically enumerates a key space to quickly find a user's key [3]. In this work we adopt the same methodologies, and provide arguments that our construction meets the necessary security requirements. We also empirically evaluate RBTs against trained forgers, generative algorithms, and our search algorithm.

3. BIOMETRICS AND KEY GENERATION

To generate a key, a user's biometric is measured as one or more digital signals, which are processed by n statistical functions, or *features* $(\phi_1, \ldots, \phi_n)$. For a biometric to be useful for key generation, the signals, and consequently the output of the features, must vary across the population. Additionally, the signals themselves can often differ between measurements of the same user. This variation is due to the natural inconsistencies inherent to human physiology or behavior. However, since cryptography requires keys to be regenerated precisely, the biometrics must be error-corrected to consistent values. Simple quantization has been shown to be useful for voice and handwriting applications [16, 22, 7].

BKGs error-correct the output of features and create cryptographic keys. This is typically accomplished with two algorithms: an *enrollment* algorithm and a *key generation* algorithm.

- $\mathsf{Enroll}(\beta_1, \ldots, \beta_\ell, \pi)$: The enroll algorithm is a probabilistic algorithm that accepts as input a number of biometric samples $(\beta_1, \ldots, \beta_\ell)$, and potentially an extra source of randomness (π), and outputs a template (T) and a cryptographic key (K). In the event that $\beta_1, \ldots, \beta_\ell$ do not meet some predetermined criteria, the enroll algorithm might output the failure symbol $\perp$.
- $\mathsf{KeyGen}(\beta, \pi, T)$: The key generation algorithm accepts as input one biometric sample (β), potentially an extra source of randomness (π), and a template (T). The algorithm outputs either a cryptographic key (K), or the failure symbol $\perp$ if the provided biometric sample cannot be used to create a key.

To see how a BKG might work in practice, consider an example where a BKG-derived key is used for laptop file encryption. Here, a user first supplies a password π and number of biometric samples to the BKG. The enrollment algorithm processes the samples to establish a "normal" profile for the user, and then outputs a template—which depends on both these samples and π—and a cryptographic key. The laptop encrypts the file with the key and stores the ciphertext, along with the template. Of course, π, the key, and the enrollment samples are then purged from the system. Later, when the user wishes to decrypt the file, she provides a password π' and a new biometric sample to the laptop, which provides these and the stored template to the key generation algorithm. If $\pi' = \pi$ and the new sample is similar to the ones provided during enrollment, then the key output by the key generation algorithm will be the same as the one used to encrypt the original file. The new key can then be used to decrypt the ciphertext file.

3.1 Security Requirements

It should be apparent from this discussion that the security of this approach relies on several assumptions. First of all, it is assumed that the user's biometric samples are not easily replicated by others. That is, the biometric should be difficult to forge, or otherwise predict using *auxiliary information*. Auxiliary information is any public information that is available to an attacker, such as other users' biometrics, templates, or keys. If this is not the case, an attacker can subvert the BKG by running KeyGen with the stored template and replicated biometric input to extract the key. It is also assumed that biometric template leaks no information about the key or the biometric. Otherwise, the attacker could use the template to help guess the key. Finally, it is assumed that the key has enough entropy to resist brute force attacks.

For completeness, we reiterate our security requirements [3]:

- *Biometric Uncertainty* (REQ-BUN): The biometric samples input to a BKG should be difficult to predict. For human ad-

versaries, this amounts to showing that realistic attackers [4] are not able to create forgeries, even when given auxiliary information. For algorithmic adversaries, this amounts to showing that the biometric has high entropy across the user population [8, 9, 3]. Also, one must show that generative algorithms cannot create accurate forgeries [4].

- *Key Randomness* (REQ-KR): Assuming REQ-BUN, the keys output by a BKG appear random to any adversary who has access to auxiliary information and the template used to derive the key. For instance, one might require that the key be computationally indistinguishable from random.

- *Strong Biometric Privacy* (REQ-SBP): Assuming biometric uncertainty, an adversary learns no useful information about a biometric given auxiliary information, the template used to derive the key, and the key itself. For instance, no computationally bounded adversary should be able to compute any function of the biometric.

In order to evaluate a practical BKG, one must show that each of these properties holds. Part of the difficulty in evaluating the security of a BKG, however, stems from the fact that both empirical and cryptographic arguments are necessary to address these requirements. More precisely, showing REQ-BUN requires an empirical argument using populations of users. On the other hand, showing REQ-KR and REQ-SBP requires cryptographic arguments showing that the template is secure provided that REQ-BUN holds. In Section 7 we argue informally that RBTs achieve REQ-KR and REQ-SBP. In Section 8 we provide an empirical evaluation of REQ-BUN for RBTs against each of the adversaries in [4] and [3].

4. OVERVIEW AND PRELIMINARIES

The most straightforward approach to designing RBTs would be to take an arbitrary template, which describes the features for the user in question, and encrypt it with a key known only to the user. Obviously, this cannot work in our setting because if we had access to a key that was strong enough for encryption then there would be no need for the BKG in the first place. Instead, we develop an approach inspired by that of Lomas et al. in a different domain, where no *verifiable* plaintexts are encrypted under the low-entropy password [14]. Specifically, the decryption of the ciphertext using any password, including the correct password, results in a high-entropy string, and so an attacker performing a brute-force search cannot tell when she has found the correct password. Similar techniques have been used by Bellovin and Merrit [5] in the context of Encrypted Key Exchange. RBTs adopt this approach and hide features by encoding templates as high-entropy strings, and encrypting these strings with low-entropy passwords.

However, in our setting, designing the template such that a decryption under the correct password is indistinguishable from decryption under an incorrect password is more challenging because templates have semantic meaning. In other words, the templates must specify the features and the error-correction information necessary to process a biometric and derive a key. Thus, we require a representation of the features and error-correction information that is random, yet meaningful. To create such a representation, we distinguish between two types of information in a template. The first type can be randomized without losing semantic meaning. We thus randomize and encrypt these pieces of information. The second type of information cannot be randomized without losing semantic meaning. We thus fix these values to be the same across the population, and include them unencrypted in the template.

RBTs use quantization for error-correction. That is, the output range of each feature is partitioned into segments of equal width, and the index of the segment that contains the feature applied to the user's samples is used for key generation. Given that our scheme uses quantization, our templates need to specify three pieces of information: the features, the offset of the quantization within the feature's output range, and the width of the quantization. We can safely encode features without losing semantic meaning if we specify features as indexes into a table, randomly assign a subset of the features to each user, and then encrypt this subset in the template. In this way, the decryption of feature indexes with an incorrect key will be indistinguishable from decryption with the correct key because in both cases, a decrypted template appears as a random permutation on a random subset of feature indexes. We can also safely encode quantization offsets. Note that if a user's quantization over the output range of feature ϕ_i is defined as the set $\{\alpha_i, \alpha_i + \delta_i, \alpha_i + 2\delta_i, \dots\}$, then knowledge of of the width of the quantization (δ_i) and any of the quantization offsets $\alpha_i + c\delta_i$ unambiguously defines the entire set. Thus, if all features have output ranges that can be mapped to some range R, then we can safely encode quantization offsets without losing semantic meaning by encrypting a random value in $\{\alpha_i, \alpha_i + \delta_i, \alpha_i + 2\delta_i, \dots\} \subseteq R$. Decryption of the resulting ciphertext under any key results in a value that is randomly distributed over R, but that is also semantically meaningful as a quantization offset for *any* feature.

On the other hand, the width of the quantization, δ_i, cannot be randomized and so we cannot safely encrypt it. To see why this is the case, note that the random assignment of a quantization width to a feature (as would happen if the template is decrypted under an incorrect password) might not be semantically meaningful. An adversary could use an observed semantic inconsistency in a decrypted template to infer that she had decrypted the template with an incorrect password. For instance, if there is a feature for which most users in the population exhibit large variation, and require large error tolerance, then an adversary who decrypted the template under an incorrect password and observed a small quantization width for that feature could logically deduce that she guessed the incorrect password. To avoid this type of problem, we specify a user-independent error-correction threshold for *each* feature.

At a high level, our construction works as follows. We encode the features by storing a table that assigns an index to each feature. The same table is stored in every template, but only the indexes that correspond to the correct features are encrypted in a particular user's template. Features are encrypted in a way such that decryption under any key specifies an index in the global table, and thus specifies a viable feature. Additionally, we ensure that the probability that any given feature is assigned to a user is the same across the population. Given this encoding algorithm, decryption of the encrypted features under any password results in a list of features that is equally likely, and so decryption under the correct password is indistinguishable from a decryption under an incorrect password.

To encode error-correction information, we follow a similar approach. First, we fix the quantization width δ_i for each feature ϕ_i, and store it in the global table. As the quantization widths are global values, we only need to encode one quantization offset to completely specify error-correction. Since any such offset suffices, we randomly select an offset and encrypt it with a pseudorandom permutation with a domain that covers all quantization widths. Decryption under any password thus results in a value that is equally likely to be a correct offset for any feature.

This approach results in an encoding algorithm where every decryption of the template simultaneously "appears random" and is useful to generate a cryptographic key. However, only the decryp-

tion under the *correct* password will yield a template that can be used to generate the *correct* key. This has the property that an adversary who searches for the key will have to guess the biometric for *each* guess at the password that was used to encrypt the template. Before presenting the technical details of the construction, we introduce some relevant cryptographic primitives and notation.

4.1 Preliminaries

We use the notation $||$ to refer to string and list concatenation, and refer to the i^{th} element in the list L as $L[i]$. We use $x \xleftarrow{R} X$ to denote the selection of an element x uniformly at random from the set X, and $x \leftarrow \mathsf{A}$ to indicate that the algorithm A outputs x. The set of integers from a to b, inclusive, is represented as $[a, b]$, and $[a, b]_k = \{a + ik \,:\, i \in [0, \lfloor (b-a)/k \rfloor]\}$.

Our construction uses several cryptographic primitives. We use pseudorandom permutations (PRPs) on sets of integers to ensure that an adversary cannot determine whether she has decrypted a template with the correct password. Let $(\mathsf{E}^D, \mathsf{D}^D)$ denote the encryption and decryption functions (a PRP and its inverse) with key space $\mathcal{K}$ and domain and range $[0, D-1]$.

We assume that users select (possibly low-entropy) passwords from a set Π, and that our BKG outputs bit-strings of length λ. Our construction uses four random oracles:

$$\mathsf{H}_{\mathsf{pass},0} : \Pi \to \mathcal{K}, \quad \mathsf{H}_{\mathsf{ver}} : \{0,1\}^* \to \{0,1\}^t$$
$$\mathsf{H}_{\mathsf{pass},1} : \Pi \to \mathcal{K}, \quad \mathsf{H}_{\mathsf{key}} : \{0,1\}^* \to \{0,1\}^\lambda$$

$\mathsf{H}_{\mathsf{pass},0}$ and $\mathsf{H}_{\mathsf{pass},1}$ map a password into different elements in the key space of the PRPs. $\mathsf{H}_{\mathsf{ver}}$ is used to generate a token to test whether the BKG has generated the correct key. $\mathsf{H}_{\mathsf{key}}$ is used to generate the final key from a user's password and biometric samples. Finally, we assume the existence of a function, $\mathsf{Permute}$, that permutes a list of numbers in a cryptographically secure sense (again, this can be implemented as a PRP).

Our design uses an ordered set of N features $\Phi = (\phi_1, \ldots, \phi_N)$, where ϕ_i is a map from the set of biometric samples to the integers $R_i = [0, r_i]$. We use quantization for error-correction. Let $\delta_i \in \mathbb{N}$ be the tolerance of the quantization for ϕ_i. This value is fixed to be the same for each user in the population. Let $\Delta = 1 + \max_i \delta_i$, quantization offsets will be encoded into the range $[0, \Delta]$ and encrypted so that decryption under any key appears as a random, yet semantically meaningful, offset.

5. CONSTRUCTION

5.1 The Enroll Algorithm

RBTs use two algorithms, Enroll (Algorithm 1) and KeyGen (Algorithm 2). The enrollment phase is a four step process: assigning features to a user, computing the necessary information to correct a user's samples to a consistent value, using the error-corrected values to create a key, and finally, encoding a secure template.

Feature Selection. To start, a user presents ℓ biometric samples $\beta_1, \ldots, \beta_\ell$ and a password $\pi \in \Pi$, to Enroll. The BKG computes some statistics over each of the samples and returns the indexes of the features that are to be used for key generation. This is a intricate process and is described in Section 6. For now, it suffices to note that from an adversary's point of view, every set of features is equally likely to be assigned to each user. Assume that the BKG assigns m features to a user. Each user is assigned a different number and/or a different set of features, so m will differ across the population. However, to ensure that all templates have the same length, Enroll pads out each template to use n features, with $m \leq n \leq N$. Let Ψ be the indexes of the m features $\phi_i \in \Phi$ that are selected for key generation, let $\tilde{\Psi}$ be the indexes of the $n - m$ features selected at random from $\Phi \backslash \Psi$ for padding, and let $L = \mathsf{Permute}(\Psi)||\mathsf{Permute}(\tilde{\Psi})$. The features specified in L, which are a random permutation of a random n element subset of Φ, are used for template creation and key generation.

Global Parameters	
Φ	The set of all features $\phi_1, \ldots, \phi_N$
Π	The set of (low-entropy) user passwords
R_i	The output range of ϕ_i, (i.e., $[0, r_i]$)
δ_i	The quantization width for ϕ_i
Δ	Maximum of quantization widths: $1 + \max_i \delta_i$
N	The number of features in Φ
n	The number of features in each template
Individual Parameters	
Ψ	Set of features assigned to a user for key gen.
$\tilde{\Psi}$	Set of features assigned to a user for padding
m	The number of features in Ψ
π	A (low-entropy) user password
β	A user's biometric reading
α_i	The smallest quantization boundary for $\phi_i(\beta)$
Cryptographic Primitives	
$(\mathsf{E}^D, \mathsf{D}^D)$	A PRP with key space $\mathcal{K}$ and domain $[0, D-1]$
$\mathsf{H}_{\mathsf{pass},0}$	Random Oracle from $\Pi \to \mathcal{K}$
$\mathsf{H}_{\mathsf{pass},1}$	Random Oracle from $\Pi \to \mathcal{K}$
$\mathsf{H}_{\mathsf{ver}}$	Random Oracle from $\{0,1\}^* \to \{0,1\}^t$
$\mathsf{H}_{\mathsf{key}}$	Random Oracle from $\{0,1\}^* \to \{0,1\}^\lambda$

Table 1: Notation

Error Correction. The next step in the enrollment process is to correct a user's samples into a single, repeatable value. RBTs use quantization for this purpose. We assume that the widths of the quantization intervals have been pre-computed (see Section 6), and are fixed across the population: feature ϕ_i uses intervals of length δ_i. To specify error-correction, the scheme need only specify a quantization offset in the range (R_i) of each feature in L. For each $i \in L$, compute the integer μ_i as the median of $\phi_i(\beta_1), \ldots, \phi_i(\beta_\ell)$. Then, partition R_i into δ_i-length intervals centered around μ_i. This requires computing one of the quantization segment boundaries, and so we compute α_i to be the smallest value in R_i that is an integer multiple of δ_i away from $\mu_i - \delta_i/2$:

$$\alpha_i = \begin{cases} \lfloor \mu_i - \delta_i/2 \rfloor \bmod \delta_i & \text{if} \quad \mu_i \geq \delta_i/2 \\ \lfloor \mu_i + \delta_i/2 \rfloor & \text{if} \quad \mu_i < \delta_i/2 \end{cases}$$

Given α_i and δ_i, the partitioning over the feature range is specified by the integers $\{0\} \cup [\alpha_i, r_i]_{\delta_i}$. The border of the partition that contains μ_i is $x_i = \max(0, \lfloor \mu_i - \delta_i/2 \rfloor)$. See Algorithm 1, lines 5–8 for the error-correction process.

Deriving a Key. Having specified our error-correction scheme, we are ready to create a cryptographic key. The key is derived from the password π, the feature indexes, and the quantized feature outputs by setting $K_j = L[j]||x_{L[j]}$ for $j \in [0, m-1]$, and setting the key to be $K = \mathsf{H}_{\mathsf{key}}(\pi||K_0|| \ldots ||K_{m-1})$ (see Algorithm 1, lines 11–12). That is, K is the output of a random oracle applied to the password, indexes of the m features selected for the user in question, and the lower boundary of the partition that contains the output of each feature. This increases the entropy over standard key generation schemes by exploiting the uncertainty associated with feature selection *in addition* to the output of each feature.

Input: The password $\pi \in \Pi$, and biometric samples $\beta_1, \ldots, \beta_\ell$
Input: (Global values): the features Φ, and quantization widths $\delta_0, \ldots, \delta_N$
Output: The key K and template T

1: $(\Psi, \tilde{\Psi}) \leftarrow \mathsf{Select}(\beta_1, \ldots, \beta_\ell)$ // Select biometric features
2: $L \leftarrow \mathsf{Permute}(\Psi)||\mathsf{Permute}(\tilde{\Psi})$
3: $k_0 \leftarrow \mathsf{H}_{\mathsf{pass},0}(\pi)$, $k_1 \leftarrow \mathsf{H}_{\mathsf{pass},1}(\pi)$
4: **for** $j \leftarrow 0$ to $|L| - 1$ **do**
5: $\quad i \leftarrow L[j]$
6: $\quad \mu_i \leftarrow \mathsf{Median}(\phi_i(\beta_1), \ldots, \phi_i(\beta_\ell))$
7: $\quad \alpha_i \leftarrow \lfloor \mu_i - \delta_i/2 \rfloor \bmod \delta_i$ if $\mu_i \geq \delta_i/2$. Otherwise, $\lfloor \mu_i + (\delta_i/2) \rfloor$.
8: $\quad x_i \leftarrow \max(0, \lfloor \mu_i - \delta_i/2 \rfloor)$ // Quantize the feature outputs
9: $\quad \gamma_i \xleftarrow{R} [\alpha_i, \Delta]_{\delta_i}$ // Select a random quantization offset
10: $\quad C_j = (\mathsf{E}^N_{k_0}(i), \mathsf{E}^\Delta_{k_1}(\gamma_i))$ // Encrypt feature index and quantization offset
11: $\quad K_j = i||x_i$
12: $K \leftarrow \mathsf{H}_{\mathsf{key}}(\pi||K_0||K_1||\ldots||K_{|\Psi|-1})$ // Derive the key
13: $C \leftarrow (C_0, C_1, \ldots, C_{|L|-1})$
14: $v \leftarrow \mathsf{H}_{\mathsf{ver}}(\pi||K_0||K_1||\ldots||K_{|\Psi|-1})$
15: **return** $K, T = (C, v)$

Algorithm 1: Specification of the RBT Enroll **algorithm**

Template Creation. Our task is to now encode the feature indexes and the quantization information so that only an individual with knowledge of π and the ability to produce a biometric that is "close" to the enrollment samples can generate the correct key (Algorithm 1, lines 9–10). To do so, we must encode L and the α_i so that they appear random, and then encrypt these values with π. We employ two PRPs, $(\mathsf{E}^N, \mathsf{D}^N)$ and $(\mathsf{E}^\Delta, \mathsf{D}^\Delta)$, to encrypt feature indexes (i.e., the $L[j]$) and the offsets in each feature range (i.e., the $\alpha_{L[j]}$). Since PRPs induce a different and equally-likely random permutation for every key, if the encoding of L and the α_i is truly random, then an adversary who decrypts the template will not be able to tell if she has done so with the correct password. We create two independent keys for each cipher from π as: $k_0 = \mathsf{H}_{\mathsf{pass},0}(\pi)$ and $k_1 = \mathsf{H}_{\mathsf{pass},1}(\pi)$, respectively. The keys are independent of one another to ensure that there is no correlation between the encryption of the feature indexes and the encryption of the quantization offsets. For each $j \in [0, n-1]$, set $i = L[j]$, and select $\gamma_i \xleftarrow{R} [\alpha_i, \Delta]_{\delta_i}$. Recall that $\Delta = 1 + \max_i \delta_i$, and so γ_i encodes α_i into a random integer that is less than the largest quantization width of any feature, and that is also an integer multiple of δ_i from α_i. The next step is to encrypt both the i and γ_i by computing $C_j = (c_{j,0}, c_{j,1}) = (\mathsf{E}^N_{k_0}(i), \mathsf{E}^\Delta_{k_1}(\gamma_i))$. Since each γ_i is encoded into the same range $[0, \Delta]$, the decryption of $c_{j,1}$ under any key results in a semantically meaningful quantization offset for *any* feature.

The template then consists of:

$$T = (C, v) = ((C_0, C_2, \ldots, C_{n-1}), \mathsf{H}_{\mathsf{ver}}(\pi||K_0||\ldots||K_{m-1}))$$

The token v is used for verification purposes, and is a function of the password and error-corrected biometric samples, but is independent of the key K because $\mathsf{H}_{\mathsf{ver}}$ and $\mathsf{H}_{\mathsf{key}}$ are independent random oracles.

5.2 The KeyGen Algorithm

The KeyGen algorithm is simpler than enrollment in that it decrypts the template, measures the biometric sample with the recovered features, and then recreates the key (see Algorithm 2). The input to the algorithm is password $\pi \in \Pi$, the template $T = (C, v)$,

Input: Template $T = (C, v)$, password $\pi \in \Pi$, the biometric sample β, the features Φ, and error-correction information $\delta_0, \ldots \delta_N$
Output: The key K, or $\perp$ on failure.

1: $k_0 \leftarrow \mathsf{H}_{\mathsf{pass},0}(\pi)$, $k_1 \leftarrow \mathsf{H}_{\mathsf{pass},1}(\pi)$
2: **for** $j \leftarrow 0$ to $|C| - 1$ **do**
3: $\quad i \leftarrow \mathsf{D}^N_{k_0}(C[j][0])$ // Extract feature index
4: $\quad \alpha_i \leftarrow \mathsf{D}^\Delta_{k_1}(C[j][1]) \bmod \delta_i$ // Extract quantization offset
5: $\quad x_i \leftarrow \max_{x \in \{0\} \cup [\alpha_i, \phi_i(\beta)]_{\delta_i}} x$ // Quantize the output of ϕ_i applied to β
6: $\quad K_j \leftarrow i||x_i$
7: $\quad$ **if** $\mathsf{H}_{\mathsf{ver}}(\pi||K_0||\ldots||K_j) = v$ **then**
8: $\quad\quad K \leftarrow \mathsf{H}_{\mathsf{key}}(\pi||K_0||\ldots||K_j)$ // Derive the key
9: $\quad\quad$ **return** K
10: **return** $\perp$ // Could not recreate the key, return failure

Algorithm 2: Specification of the RBT KeyGen **algorithm**

and a biometric sample β. First, KeyGen derives the decryption keys $k_0 = \mathsf{H}_{\mathsf{pass},0}(\pi)$ and $k_1 = \mathsf{H}_{\mathsf{pass},1}(\pi)$ and uses these keys to decrypt the template and recreate the list L and the quantization offsets (i.e., the $\gamma_{L[j]}$) (see Algorithm 2, lines 3–4). For $j \in [0, |C|-1]$, let $(L[j], \gamma_{L[j]}) \leftarrow (\mathsf{D}^N_{k_0}(c_{j,0}), \mathsf{D}^\Delta_{k_1}(c_{j,1}))$. The values in L are a list of indexes that specify features. Let $i = L[j]$ be such an index. Then γ_i is the encoding of α_i, which specifies the offset of the quantization for ϕ_i. Thus, the partitioning boundaries over $[0, r_i]$ can be recreated as $\{0\} \cup [\gamma_i \bmod \delta_i, r_i]_{\delta_i}$.

To recreate the key, KeyGen measures β and sets x_i to be the largest partition boundary that is less than or equal to $\phi_i(\beta)$ (see Algorithm 2, line 5). Then, letting $K_j = i||x_i$, the algorithm iteratively attempts to recreate the key by checking that $v = \mathsf{H}_{\mathsf{ver}}(\pi||K_0||K_1||\ldots||K_j)$ for $j \in [0, |C|-1]$. If this is the case for any j, then KeyGen outputs the key $K = \mathsf{H}_{\mathsf{key}}(\pi||K_0||K_1||\ldots||K_j)$. Otherwise, the algorithm has failed and it outputs the failure symbol $\perp$. This iterative reconstruction process is necessary because the key is only derived from the features in Ψ, and it is impossible to determine where these features end, and where the extra features that were added as padding from $\tilde{\Psi}$ begin.

5.3 Correctness

Correctness amounts to showing that if $\mathsf{Enroll}(\beta_1, \ldots \beta_\ell, \pi)$ outputs $T = (C, v)$, and K, and $\mathsf{KeyGen}(\beta', \pi', T)$ outputs K', then if $\pi = \pi'$ and β' is close to the median of $\beta_1, \ldots, \beta_\ell$, then $K = K'$. For this to happen two properties must hold: first, KeyGen must use the same features as Enroll and quantize the output range of each feature in the same way as Enroll. Second, this quantization must map β' to the same segment as the μ_i that was computed from $\beta_1, \ldots, \beta_\ell$ by Enroll. If these two properties hold, then the input to $\mathsf{H}_{\mathsf{key}}$ (i.e., $\pi||K_0||\ldots||K_{|\Phi|-1}$) will be the same for both KeyGen and Enroll, and consequently both algorithms will output the same key.

It follows from the correctness of PRPs that if $\pi = \pi'$, KeyGen will correctly decrypt C to extract the i and the γ_i that were encoded by Enroll. This implies that both algorithms will use the same set of features. To see that they quantize the output range of each feature in the same way, observe that $\gamma_i \equiv \alpha_i \mod \delta_i$. Since δ_i is publicly known, KeyGen can reliably extract α_i, which is a boundary of one of the segments in the range of ϕ_i. Thus, KeyGen can recover the quantization of the range of ϕ_i as $\{0\} \cup [\alpha_i, r_i]_{\delta_i}$; and this is the same set that was used by Enroll.

Now that we have established that both algorithms process the biometric input in the same way, we show that if β' is close to the

median of $\beta_1, \ldots, \beta_\ell$, then $K = K'$. Recall that $K \leftarrow \mathsf{H}(\pi \mid\mid K_0 \mid\mid \ldots \mid\mid K_{|\Psi|-1})$, with $K_j = i||x_i$. The only part of the input that is computed from biometric input is x_i, which is the lower bound of the δ_i-length segment that contains $\mu_i = \mathsf{Median}(\phi_i(\beta_1), \ldots, \phi(\beta_\ell))$. If β' is close to the samples that were provided during enrollment, then $|\phi_i(\beta') - \mu_i| < \delta_i/2$. This implies that $\phi_i(\beta')$ falls within the same segment as the original median, and so x'_i, the lower boundary computed by KeyGen, will be the same as the x_i computed by Enroll. Thus, since both algorithms compute $\mathsf{H}_{\mathsf{key}}$ over the same input string, both will output the same key.

6. FEATURE SELECTION

The strength of any BKG is defined by the features used to measure the biometric inputs. In the case of RBTs, feature selection is especially important. We need to ensure that two properties hold. First, we must ensure that feature assignment (i.e., the output of Select) appears random to an adversary. The security of our construction rests on this property. To do so, we ensure that each feature is assigned to a user with an independent and uniform probability (Section 6.1). Second, we must select features that are resilient to forgeries, and that have high entropy across the population. This amounts to selecting features that have high entropy with uncorrelated outputs, and that are difficult to forge (Section 6.1). Using such features increases the strength of RBTs to human forgers, generative algorithms, and search algorithms.

6.1 Selection as a Random Permutation

Security of RBTs hinges upon the fact that decryption of a template under the correct password has the same distribution as decryption under an incorrect password. Decryption of $c_{0,0}, \ldots, c_{n-1,0}$ (i.e., the encryption of the indexes in L) under an incorrect password results in a random permutation on a randomly selected subset of the feature indexes. This implies that the feature indexes in templates (i.e., L) must also appear to be a random permutation on a randomly selected subset of feature indexes. Randomly permuting a set of indexes is simple, it is more challenging to select a random subset to permute. We cannot simply randomly assign features to each user; there is no indication that a randomly selected feature would be useful for a specific user. Instead, we would like to assign users those features for which they exhibit low variance, which reduces False Reject Rates and False Accept Rates, and increases entropy.

Outwardly, the requirement of selecting strong features and the requirement of selecting random features appear to be at odds with one another. However, one can actually use the selection of strong features to *force* a uniform distribution, at least, from the point of view of an adversary, over the selection process. Observe that in order for a feature to be strong for a specific user, the user will require very little error-correction to reliably repeat the output of that feature. In fact, a user will require less error-correction for the feature than other users for which the feature is less useful. Suppose the user u requires quantization with tolerance of $d_{u,i}$ to reliably correct a feature ϕ_i. If we assign feature ϕ_i to u only if $d_{u,i}$ is in the smallest k^{th} percent across the population, and fix k across all features, then we immediately have a technique that evenly distributes feature assignment across the population, in that feature ϕ_i is assigned to exactly k percent of the users. Moreover, if we comprise Φ of features that vary independently of one another, and hence the event that $d_{u,i}$ is in the smallest k^{th} percentile is independent of the event that $d_{u,j}$ is in the smallest k^{th} percentile, then each feature is assigned to a user with an independent and identical probability.

Independence in Feature Selection. The first step of ensuring that feature assignment results in a random permutation on a randomly selected subset of Φ is to ensure that the probability that feature ϕ_i is assigned to a user is independent of the probability that feature ϕ_j is assigned to that user. Since features are assigned to users based on the user's ability to consistently replicate that feature, we need to compose Φ of features such that the degree of variation is independent across features. Technically, we should ensure that every distinct subset of features is assigned to each user independently. However, due to computational and data restrictions, both on our part and the part of potential adversaries, this is infeasible and so our technique only ensures that individual features are assigned with independent probabilities.

To do so, we first generate a large set of features $\hat{\Phi}$, and conduct an empirical analysis to determine which features' error-correction tolerances (i.e., the δ_i) are correlated with one another. We suggest using a greedy algorithm to remove those features that are correlated with many other features. Such an analysis is performed with a fixed error-correction percentile k and a fixed population of users U. For each user $u \in U$, we compute the minimum error-correction threshold $d_{u,i}$ required to reliably correct feature $\phi_i \in \hat{\Phi}$. Let I_i be a binary random variable associated with feature ϕ_i. The distribution for I_i is estimated from a population of users: for a user $u \in U$, $I_i = 1$ if $d_{u,i}$ is in the smallest k^{th} percentile across U, and 0 otherwise. To remove the correlated features, we compute the statistical correlation ρ_{I_i,I_j} between all I_i and I_j and use a greedy algorithm to remove those features that have a correlation $|\rho_{I_i,I_j}| > \tau_{\mathsf{var}}$, where τ_{var} is a tunable parameter. The remaining set of features comprise Φ. If U is chosen to be large enough to be representative of the population of users who use the system, then this process need only be performed once.

Uniformity in Feature Selection. Having ensured that the features in Φ are assigned to a user with pairwise independent probabilities, the task at hand is to also ensure that these probabilities are uniform. We do this by empirically selecting δ_i such that k percent of the users require error tolerance less than δ_i. This is accomplished during enrollment with the Select algorithm, which cycles through each feature and determines from the enrollment samples how much tolerance is needed to correct all of the feature values to one segment. If this tolerance is less than the global threshold δ_i, then the feature is assigned to the user. (See the full version of this paper [2] for more details on Select.) As such, the user is assigned only those features that she can repeat consistently. Since δ_i is small, adversaries should have a greater difficulty replicating each feature to within that tolerance, and the output range of the feature is partitioned into more segments, yielding a potentially greater search space for the attacker. Our experimental results in Section 8 indicate that this is indeed the case.

Practical Considerations. While we have provided techniques to assign features to users in a way that meets our cryptographic requirements, there are several other practical considerations that must be addressed. First, we must consider the composition of Φ. Clearly, RBTs benefit from large feature sets. Since there are $\binom{N}{n}$ equally likely templates, larger feature sets imply greater uncertainty for an adversary. At the same time, however, since we assume that the size of the password space might be small, we must also endeavor to find features that resist forgery and searching attacks. That is, we cannot simply add many random features to Φ. There are also other factors that govern how we craft Φ. To ensure that the derived keys have high entropy, each of the features in Φ should have uncorrelated outputs. This reduces the likelihood of success of the search attacks similar to those described in [3]. All

of these constraints reduces the set of possible features that can be used to create Φ.

There is also the matter of selecting k, the value that determines the percentile that will be used for error-correction. There is a trade off between the number of features assigned to each user (for large values of k) and the resistance to forgability (smaller values of k). Since it is difficult to optimize feature selection across the constraints imposed by entropy requirements, forgery resiliency, correlation between features, and the selection of k, we adopt the following iterative approach. First, we create a large set of features. Then, we compute the statistical correlation over the outputs and variation of each feature. We use a greedy algorithm to remove those features that have high correlation with many other features. We then perform an empirical evaluation to measure the entropy and False Accept Rates over each feature. Then, Φ is composed of the union of two distinct sets: the features with the highest entropy, and the features with the smallest FARs. The goal is that by combining both sets, RBTs will be resilient to both forgeries and search algorithms.

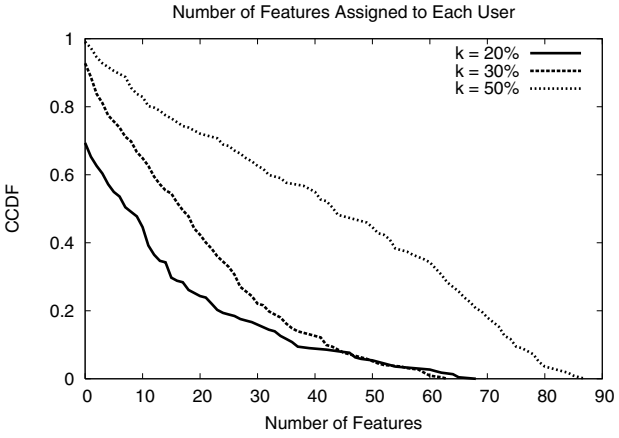

Figure 1: The number of features assigned to each user.

6.2 Feature Selection Evaluation

In this section we provide empirical evidence that Select acts as a random permutation on an n element subset of Φ. In order to provide a concrete analysis, we focus on one biometric modality: handwriting. Our results are based on the data set described in [4] that consists of over 9,000 writing samples from 47 users. Each user provided 10-20 enrollment samples for five different phrases. We emphasize that these phrases are used simply to extract biometric readings; they are unrelated to the low-entropy password π used by RBTs. The data set also consists of a number of forgeries, we only use the stronger "trained" forgeries for our security analysis. The data set also contains approximately 3,000 phrases that are used to generate a "parallel corpus" to drive generative algorithms.

In this section we analyze our feature selection strategy to study the impact of selecting different error-correction percentiles (i.e., k). Recall that the quantization widths used to error-correct each feature are fixed across the population, and the level of quantization is specified by the value k. For large values of k, users will be assigned more features, but the features will be easier to forge and will have lower maximum theoretical entropy. For small values of k, users will be assigned fewer features, but they will be more difficult to forge and will have higher maximum theoretical entropy. To explore this tradeoff, we performed the feature selection process for $k = 20\%$, $k = 30\%$, and $k = 50\%$.

For $k = 20\%, 30\%, 50\%$, with $\tau_{\text{var}} = .6$, our feature assignment process resulted in feature sets of size $N = 69, 72, 86$, respectively. Figure 1 shows a CCDF of the number of features that were assigned to each user. As is to be expected, the Select algorithm assigns more features to each user for larger values of k. In fact, for $k = 50\%$, fewer than 1% of users were assigned no features, 80% of users were assigned at least 12 features and 50% of the users were assigned more than 43 features. The results for $k = 20\%$ are not as encouraging: 30.6% of the population were not assigned any features. This implies that setting $k = 20\%$ might not yield a BKG that is useful for a large percentage of the population.

We also performed an analysis to demonstrate that our feature selection technique outputs a random subset Φ. We focused on a BKG with $k = 30\%$, and the maximum number of features in each template was set to $n = 50$. For each feature in Φ we computed the probability that a given feature ϕ_i appears in a template, and the conditional probability that ϕ_i appears in a template given that $\phi_j \neq \phi_i$ also appears in that template. We measured the observed probabilities and conditional probabilities. If the selection process results in a random permutation on a random subset of Φ, then we would expect ϕ_i to appear in a template with probability 0.694. In this case, we would also expect the conditional probability that ϕ_i and ϕ_j both appear in a template to be 0.69. Indeed, our results are inline with this expectation. In fact, we performed the χ^2-Test of Homogeneity [17] to ensure that the observed probability distribution is indeed uniform, and the χ^2-Test of Independence [17] to ensure that the likelihood that two features appear in a template are independent. In both cases, we would accept the hypothesis that feature selection acts as a random permutation on a random subset with confidence level $\alpha = .99$. (See the full version of this paper [2] for further details on our testing methodology and results.) We argue that this provides empirical evidence that our feature assignment algorithm achieves the goals that are required to meet the necessary security properties.

7. SECURITY SKETCH

In this section we provide informal arguments as to why RBTs are secure. That is, we provide evidence that given access to auxiliary information, the template, and the key, an adversary cannot learn significant information about the biometric (REQ-SBP); and that an adversary cannot use auxiliary information and a biometric template to distinguish the correct key from random (REQ-KR). Our arguments are based on the assumption that feature selection acts as a random permutation on a random n-element subset of Φ, which we have argued, both constructively (Section 6.1) and empirically (Section 6.2).

7.1 Strong Biometric Privacy

Our conjecture is that since feature assignment results in a random permutation of a random n-element subset of Φ, the optimal strategy for an attacker is to simultaneously enumerate Π and the set of all biometrics until she derives the correct key. While she can use the template and key to verify that a guessed biometric sample/password pair is correct, the attacker cannot use information gleaned from the template or key to create a guess of the biometric.

Our reasoning for this argument is as follows: assume that an adversary has access to a template $T = (C, v)$ and K. Note that since v and the K are the output of public functions, an attacker can guess a biometric input and a password, run KeyGen with T and check that the output matches K to test the correctness of her guesses. Aside from this, since v and K are the output of random

oracles, they cannot be used to determine the values of the enrollment samples.

We now turn our attention to the encoding of the feature indexes and quantization information, $C = ((c_{0,0}, c_{0,1}), (c_{1,0}, c_{1,1}), \ldots, (c_{n-1,0}, c_{n-1,1}))$. Since each element in C is encrypted under a potentially low-entropy password, one must show that the entropy of the plaintext is high, and so an adversary who guesses a correct password cannot distinguish the original plaintext from a decryption of C under an incorrect password. If this is the case, then an adversary who guesses passwords must enumerate the set of biometric samples for each guess to determine whether her guesses are correct. The point is to show that such an attacker does not glean any information about the biometric sample from the template, but rather she simply enumerates samples through auxiliary means. If true, this implies that REQ-SBP holds.

We now argue the aforementioned point informally as follows. Let L be the list of indexes that Enroll assigns to the target user, and $G = (\gamma_{L[0]}, \gamma_{L[1]}, \ldots, \gamma_{L[n-1]})$ be the randomized encoding of the quantization offsets for each feature in L. Assume that Enroll uses the two keys $k_0 \leftarrow \mathsf{H}_{\mathsf{pass},0}(\pi)$ and $k_1 \leftarrow \mathsf{H}_{\mathsf{pass},1}(\pi)$ to encrypt each of these lists to create the list C. Adopting the notation that for a function F and a list $X = (x_1, \ldots, x_m)$, that $\mathsf{F}(X) = (\mathsf{F}(x_1), \ldots, \mathsf{F}(x_m))$, we decompose C into two parts:

$$C_0 = (c_{0,0}, c_{1,0}, \ldots, c_{n-1,0}) = \mathsf{E}^N_{k_0}(L)$$
$$C_1 = (c_{0,1}, c_{1,1}, \ldots, c_{n-1,1}) = \mathsf{E}^{\Delta}_{k_1}(G)$$

Our goal is to argue that for all $(k'_0, k'_1) \leftarrow (\mathsf{H}_{\mathsf{pass},0}(\pi'), \mathsf{H}_{\mathsf{pass},1}(\pi'))$, L is indistinguishable from $\mathsf{D}^N_{k'_0}(C_0)$ and G is indistinguishable from $\mathsf{D}^{\Delta}_{k'_1}(C_1)$.

Note that decryption of C_0 or C_1 with an incorrect key results in a random permutation on a random n element subset of the corresponding domain. Since Select assigns a random n element subset of Φ to each user and the Enroll algorithm randomly permutes this set to create L, L is a random permutation of a random n element subset of Φ. Thus, L is indistinguishable from $\mathsf{D}^N_{k'_0}(C_0)$.

Lastly, to see why G is indistinguishable from $\mathsf{D}^{\Delta}_{k'_1}(C_1)$, observe that Enroll creates γ_i by selecting a random element in $[0, \Delta]$ that is an integer multiple of δ_i from the border of the smallest quantization offset (Algorithm 1, lines 7 and 9), which is computed from the median of the enrollment samples. Since an attacker does not know this precise median[1] γ_i is also difficult to predict. Thus, γ_i is randomly distributed over $[0, \Delta]$, and so G is a list of random values over $[0, \Delta]$. Note that $\mathsf{D}^{\Delta}_{k'_1}(C_1)$, is also a list of random values over $[0, \Delta]$. Since L is indistinguishable from $\mathsf{D}^N_{k'_0}(C_0)$ and G is indistinguishable from $\mathsf{D}^{\Delta}_{k'_1}(C_1)$, we argue that REQ-SBP holds.

7.2 Key Randomness

Next, we argue that an adversary cannot use a template, $T = (C, v)$, to distinguish K from random—i.e., REQ-KR. In our case this amounts to arguing two properties: C cannot be used to infer the inputs to $\mathsf{H}_{\mathsf{key}}$ (i.e., π, the indexes of the features, and the quantization offset that contains the user's sample), and that v cannot be used to distinguish K from random. To argue that the first property holds, we follow the same argument that we used to argue REQ-SBP. There, we argued that the best strategy that an attacker has to find the indexes and quantization information is to simultaneously enumerate both the password space and biometric space to find the values that were provided to Enroll. This implies that if the combined entropy of both the password space and biometric space are sufficiently high, then an attacker cannot infer the input to $\mathsf{H}_{\mathsf{key}}$.

To argue that v is of no value to an attacker, we note that although v and K are derived from the same input, they are the outputs of two independent random oracles. Thus, v leaks no information about K.

[1]We assume that for most "normal" biometrics, this median is uniformly distributed. We believe this to be a reasonable assumption as legitimate users cannot even precisely recreate this value.

8. EMPIRICAL STUDY

In this section we evaluate our proposal using a host of recently-proposed techniques. We examine the resiliency to forgery against of trained forgers [4]; study the impact of generative algorithms [4]; and the test resiliency against the search algorithm described in [3].

Since the goals of this section are to evaluate the strength of the features that RBTs use, our analyses assume that an adversary has access to the correct password π. That is, the attacker can correctly decrypt the template and so her task is only to guess the biometric input. This implies that the results presented in this section are a *lower bound* on the security that the scheme would provide in practice. Nonetheless, this is an important viewpoint, as it allows evaluators to better understand how much extra security the biometric adds to the strength of the password.

8.1 Resistance to Forgeries

First, we show that RBTs can withstand forgeries from trained human forgers and generative algorithms. We use the same data set that was used to analyze feature selection, and apply our techniques to unencrypted RBT-generated templates. We present results for RBTs based on two choices of k, and the maximum set of features that can be encoded into each template (n): one with $k = 30\%$ and $n = 50$, and the other with $k = 50\%$ and $n = N$.

We generate templates for each user for each combination of k and n. To compute the FRR we use repeated leave-out-κ cross validation. Given ν enrollment samples, we randomly choose $\nu - \kappa$ samples to select features and create a template. Then, we use the remaining κ samples to create keys with the template, and measure the number of features that are not successfully recreated. We set ν and κ to be in the ratio of 3:1 and use all samples in the data set. This process is repeated 10 times and averages across all 10 runs are used to compute the FRR.

To compute the FAR, we use all of the user's samples to create a template and key, and test the ability of forgers to recreate the correct key. We only report FRR and FAR for those users who were assigned a minimum number of features during enrollment. We classified other users as failing to enroll. For $k = 30\%$ and $k = 50\%$, the minimum number of features required for enrollment was set to 4 and 10, respectively.

We evaluated the strength of RBTs using new evaluation methodologies [4]. Specifically, we focus on trained human forgers [4], and the Concatenative Synthesis (CS) generative algorithm [4]. Recall that trained forgers are human forgers whose natural ability to create forgeries has been improved through training and motivation. As in previous experiments, we provide trained forgers with a real-time rendering of the target user's passphrase during the forging process. The generative algorithms use limited information form a target user along with population statistics to create forgeries. Specifically, CS combines real-time samples from a target user with general population statistics to create forgeries.

Table 2 provides the EERs for RBTs as well as the number of errors that the BKG must correct after quantization to generate the correct key. EERs are provided for trained forgers and Concatenative Synthesis, against two different RBTs and against a the "Baseline" quantization-based BKG of Vielhauer and Steinmetz [22] under the strengthened 36 feature set described in [4]. We compare

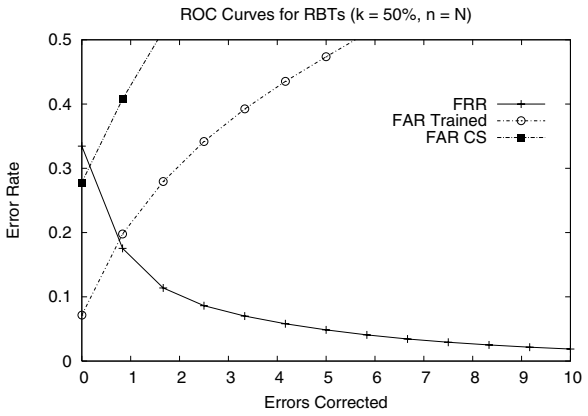

Figure 2: Performance of RBTs with $k = 50\%$ and $n = N$. Forgeries are taken from trained forgers, and the Concatenative Synthesis algorithm.

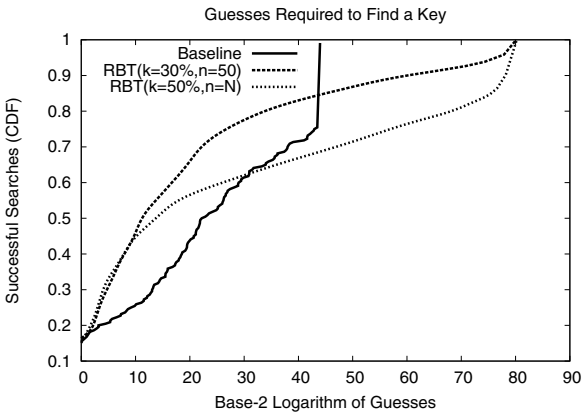

Figure 3: The number of guesses required by the search algorithm to find RBT-derived keys. We compare the RBTs to a Baseline approach that is designed to resist searches.

	RBT (a)		RBT (b)		Baseline	
	EER	Errors	EER	Errors	EER	Errors
Forgers	17.7%	1	19%	1	20.6%	4
CS	27%	1	30%	0	27.4%	3

Table 2: Comparison between RBTs and a baseline approach. RBT (a) operated with parameters ($k = 30\%$, $n = 50$) and RBT (b) operated with parameters ($k = 50\%$, $n = N$).

our approach to this Baseline because we believe it to be a state-of-the-art handwriting-based BKG. However, it is not possible to directly compare the overall security afforded by both constructions because RBTs also use human supplied passwords. Nonetheless, since our security analysis assumes that an attacker has access to π, we are able to compare the techniques that both approaches use to to extract entropy from biometric inputs.

As Table 2 shows, for each BKG, the strongest adversary is the Concatenative Synthesis algorithm. For both RBTs, the EER under this type of attacker is approximately 28.5% at no errors corrected by the BKG. Although trained human forgers have access to the entire biometric when creating their forgeries, they do not perform as well as CS, achieving an EER of approximately 18% at 1 error-corrected for both choices of parameters. The ROC curve in Figures 2 provides a more detailed point of view of the performance of RBTs against each of the forgers.

Discussion. These results lead to several observations. Under the assumption that the attacker has access to π, RBTs are comparable to the standard approach against trained human forgers and CS. However, we observe that RBTs increase the difficulty for each type of attacker, because in addition to collecting writing samples, an attacker must also garner the correct password.

Another important takeaway is where the EERs occur. This indicates how much extra work the BKG must perform to generate a correct key for legitimate users. An EER that occurs at x errors corrected requires the BKG to perform a search to identify the features that the user has incorrectly recreated. Assuming a template length of n, and that each quantized feature takes on at most y values, this could be a space of up to $\sum_{i=0}^{x} y^i \binom{n}{i}$. For reasonable values of y and n, and any value of $x > 2$, this becomes computationally prohibitive, rendering the construction infeasible in practice.

The EER for RBTs always occurs at 0 or 1 errors, whereas, as is the case with many prior constructions, the EERs for the Baseline occur at 3-4 errors. Thus, RBTs offer more resistance to forgery against each of type of attacker, and do so in a more practical manner. We hypothesize that this is because RBTs allow us to employ stronger features. For a detailed description of the features, see the full version of this paper [2]. Since RBTs employ stronger features, they allow for more error tolerance for each feature, while maintaing security. This causes the FRR of RBTs to drop quickly. Observe that at one error corrected, the RBTs with $(k = 30\%, n = 50)$ and $(k = 50\%, n = N)$ had FRRs of 19.4% and 16%, respectively. By contrast, at one error-corrected, our implementation of the Baseline had a FRR of 57%. If we tune the tolerances so that the Baseline achieved similar FRRs, the FAR would likely also increase.

8.2 Resistance to Searches

Having explored the ability of RBTs to withstand forgeries, we now focus on their ability to resist search algorithms. Instead of estimating Shannon entropy, we empirically and estimate the number of guesses that an adversary would require to find a key. This approach is more feasible in light of real-world data constraints. In particular, we estimate REQ-BUN against algorithms by applying the search algorithm described in [3] to RBT-generated templates and keys. We use only those users who successfully enrolled during our forgery experiments, and again tested the system with the parameters $(k = 30\%, n = 50)$ and $(k = 50\%, n = N)$. As before, we assume that the attacker has access to π, and so the results we present represent a worst-case point of view.

The results of the tests are shown in Figure 3. Again, we also plot the results from another Baseline that is based on Vielhauer and Steinmetz [22], but utilizes a different set of features that are selected to resist searching attacks. Overall, even with access to the target user's password π, the RBT with parameters $(k = 50\%, n = N)$ is stronger than the baseline approach for 40% of the population; for over 30% of the population, this improvement is dramatic. To be fair, we note that, unfortunately, both instantiations of the RBT suffer the same difficulty as the Baseline for a non-trivial percent of the population; for $\approx 15\%$ of the users, the algorithm correctly predicts the target user's key on the first attempt—which underscores the power of the original search-based attack [3]. The Baseline then outperforms both variants of the RBTs up until the

60^{th} percentile across the population. After this point, however, both variants of the RBTs improve, while the baseline construction remains relatively constant. At 2^{44} guesses the search algorithm is able to exhaust the complete key space of the Baseline. However, this success rate does not happen for either of the RBT-based constructions until 2^{80} guesses. Indeed, the algorithm required $\approx 2^{80}$ guesses for about 14.6% of the population.

9. CONCLUSION

We propose a new technique to extract more entropy from biometrics for key generation purposes. A key idea in this work is to measure biometrics differently for each user. We argue that doing so offers several advantages over classic techniques: Since we assign only strong features to each user, we are able to reduce error rates, reduce susceptibility to forgery, and increase the maximum potential theoretical entropy that can be extracted from the biometric measurement. More importantly, in addition to guessing the biometric sample that was used for key generation, adversaries must now also guess how this sample is measured. We analyze our construction with recently proposed evaluation techniques and also provide informal security arguments as to why our construction is secure. We show that RBTs allow many users to generate keys that are stronger than passwords alone.

Acknowledgments

We would like to thank Dan Lopresti for helpful discussions throughout the course of this work. We would also like to thank the anonymous reviewers for their comments. This work was funded in part by NSF Grant CNS-0430338 and the Phillips and Camille Bradford Fellowship.

10. REFERENCES

[1] A. Alvare. How Crackers Crack Passwords or what Passwords to Avoid. In *Proceedings of the Second USENIX Security Workshop*, pages 103–112, August 1990.

[2] L. Ballard. *Robust Techniques for Evaluating Biometric Cryptographic Key Generators*. PhD thesis, The Johns Hopkins University, 2008. Available at `http://www.cs.jhu.edu/~lucas/papers/thesis.html`.

[3] L. Ballard, S. Kamara, and M. K. Reiter. The Practical Subtleties of Biometric Key Generation. In *Proceedings of the 17^{th} Annual USENIX Security Symposium*, pages 61–74, San Jose, CA, August 2008.

[4] L. Ballard, F. Monrose, and D. Lopresti. Biometric Authentication Revisited: Understanding the Impact of Wolves in Sheep's Clothing. In *Proceedings of the 15^{th} Annual USENIX Security Symposium*, pages 29–41, Vancouver, BC, Canada, August 2006.

[5] S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 72–84, Washington, DC, USA, 1992. IEEE Computer Society.

[6] X. Boyen. Reusable Cryptographic Fuzzy Extractors. In *ACM Conference on Computer and Communications Security*, pages 82–91. ACM, 2004.

[7] Y.-J. Chang, W. Zhang, and T. Chen. Biometrics-Based Cryptographic Key Generation. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, volume 3, pages 2203–2206, 2004.

[8] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In *Proceedings of Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer-Verlag, 2004.

[9] Y. Dodis and A. Smith. Correcting Errors Without Leaking Partial Information. In *Proceedings of the 37^{th} ACM Symposium on Theory of Computing*, pages 654–663. ACM, 2005.

[10] D. Feldmeier and P. Karn. UNIX Password Security – Ten Years Later. In *Proceedings of Advances in Cryptology – CRYPTO*, pages 44–63, 1990.

[11] F. Hao, R. Anderson, and J. Daugman. Combining Crypto with Biometrics Effectively. *IEEE Transactions on Computers*, 55(9):1081–1088, September 2006.

[12] A. Juels and M. Sudan. A Fuzzy Vault Scheme. *Design, Codes and Cryptography*, 38(2):237–257, 2006.

[13] A. Juels and M. Wattenberg. A Fuzzy Commitment Scheme. In *Proceedings of the Sixth ACM Conference on Computer and Communication Security*, pages 28–36, November 1999.

[14] T. M. A. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham. Reducing risks from poorly chosen keys. In *Proceedings of the 12th ACM Symposium on Operating System Principles*, pages 14–18, Dec. 1989.

[15] F. Monrose, M. K. Reiter, Q. Li, and S. Wetzel. Cryptographic Key Generation from Voice (Extended Abstract). In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 12–25, May 2001.

[16] F. Monrose, M. K. Reiter, and S. Wetzel. Password Hardening based on Keystroke Dynamics. *International Journal of Information Security*, 1(2):69–83, February 2002.

[17] J. A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, Belmont, CA, 2nd edition, 1995.

[18] C. Soutar, D. Roberge, A. Stoianov, R. Gilroy, and B. V. Kumar. Biometric Encryption$^{\text{TM}}$ using Image Processing. In *Optical Security and Counterfeit Deterrence Techniques II*, volume 3314, pages 178–188. IS&T/SPIE, 1998.

[19] J. Thorpe and P. van Oorschot. Human-Seeded Attacks and Exploiting Hot-Spots in Graphical Passwords. In *Proceedings of the 16^{th} Annual USENIX Security Symposium*, Boston, MA, August 2007.

[20] U. Uludag, S. Pankanti, S. Prabhakar, and A. K. Jain. Biometric Cryptosystems: Issues and Challenges. *Proceedings of the IEEE Special Issue on Multimedia Security of Digital Rights Management*, 92(6):948–960, 2004.

[21] P. van Oorschot and J. Thorpe. On Predictive Models and User-Drawn Graphical Passwords. *ACM Transactions on Information and System Security*, 10(4):1–33, 2007.

[22] C. Vielhauer and R. Steinmetz. Handwriting: Feature Correlation Analysis for Biometric Hashes. *EURASIP Journal on Applied Signal Processing*, 4:542–558, 2004.