Foreword

This chapter is based on lecture notes from coding theory courses taught by Venkatesan Guruswami at University at Washington and CMU; by Atri Rudra at University at Buffalo, SUNY and by Madhu Sudan at MIT.

This version is dated **April 1, 2013**. For the latest version, please go to

http://www.cse.buffalo.edu/ atri/courses/coding-theory/book/

The material in this chapter is supported in part by the National Science Foundation under CAREER grant CCF-0844796. Any opinions, findings and conclusions or recomendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Chapter 16

Securing Your Fingerprints: Fuzzy Vaults

String-based passwords are the dominant mode of authentication in today's information-reliant world. Indeed, all of us use passwords on a regular basis to authenticate ourselves in almost any online activity. Strings have become widespread due to several nice mathematical properties. First, matching two strings (that is, checking if two strings are exactly the same) is computationally very fast (and easy). Second, and more importantly, there exist secure hash functions that map a string x to another string $h(x)$ such that, given $h(x)$, determining x is hard. Furthermore, since $h(x)$ is itself a string, we can check if a claimed password y is the same as the original string x by comparing $h(y)$ and $h(x)$, which (as we just observed) is easy to do. This implies that the server performing the authentification only needs to store the hashes $h(x)$ of the original passwords. Hence, even if the list of hashed passwords were compromised, the passwords themselves would remain secure.

The above scheme is perfect as long as the passwords x are "random enough," and this can be achieved if the passwords were generated randomly by some automated process. However, in real life passwords are generated by humans and are not really random. (One of the most quoted facts is that the most commonly used password is the string "password" itself.) Further, we tend to forget passwords, which has lead to the near ubiquity of "Forgot passwords" links in places where we need to login.

One alternative that has been gaining traction in the last decade or so is to use a user's fingerprint as their password. The big advantage is that it is hard to "forget" one's fingerprint. In this chapter, we will look at the issues in using fingerprints as passwords and see how Reed-Solomon codes can help.

16.1 Some quick background on fingerprints

For the time being let us assume that we can somehow convert a fingerprint (image) somehow to a string f. (We will see more details on this shortly.) Then we have the following naive solution:

Naive Solution. Use any off-the-shelf hash function h for strings and then store $h(f)$ instead of f.

To see the issues with the naive solution, we first need to know a little bit about how fingerprints are stored. The standard way to store a fingerprint is as a collection of triples, called minutia. Each minutia point is located where one ridge splits into two, or where one ridge ends. The i^{th} minutia is the triple (x_i, y_i, θ_i), where x_i and y_i are the x and y coordinates of a point on the finger, and θ_i indicates the direction of the ridge that created the minutia point relative to the plane.

The main issue with our naive solution is that two fingerprint readings will never be exactly the same, even if the same finger is used. For any two fingerprint readings, the following issues may produce errors:

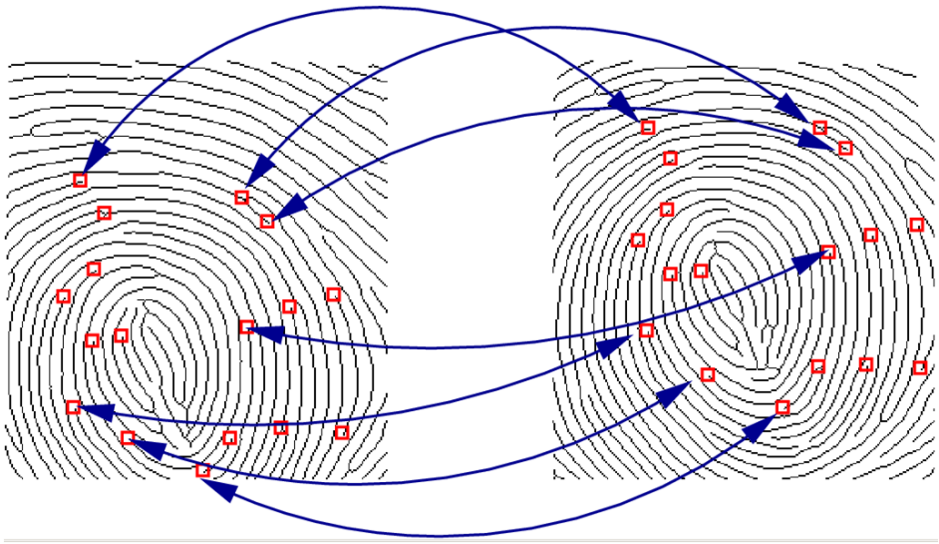
1. Translation and rotation, even when using the same finger.

2. Varying pressure.

3. Each reading may not completely overlap with the ideal fingerprint region (i.e., the finger may be slightly tilted).

4. The minutia are not ordered, so they form a set instead of a vector. Of course one can sort the set to produce a string, but in conjunction with the earlier issue (especially those involving rotation and translation) this implies that the values of (x_i, y_i, θ_i) by themselves are not that important. Furthermore the fact that two readings might not have complete overlap means that we are interested in matching readings that have significant overlap, so it turns out that the set notation is deal to theoretically deal with the issues.

Figure 16.1: The minutia are unordered and form a set, not a vector.



We can now see that the naive solution is inadequate. Even if we could somehow correct the first three issues, existing hash functions for strings require a vector, not a set, so our naive solution will fail.

Remark 16.1.1. The four problems that came up in our naive solution will come up in any solution we propose. Technology has not yet developed to the point where we can securely eliminate these issues, which is why there are no prevalent secure commercial systems that safeguard secrets using fingerprints. (The reason government agencies, such as the police or FBI, use fingerprinting is because there is an inherent trust that the government will keep your data secure, even when it does not apply a good hash function to it.)

Thus, what we are looking for are secure hash functions designed to handle the additional challenges posed by fingerprints. We would like to mention that for fingerprints to replace strings as passwords, the hash function needs to satisfy both of these properties *simultaneously*: (i) we should be able to match hashes from the "same" fingerprint and (ii) an adversary should not be able to "break" the hash function.

16.2 The Fuzzy Vault Problem

We begin with the fuzzy vault problem, which is slightly different from the one we have been studying so far. Say you have a secret string, s, and you want to store it in a secure way. Instead of using a password, you want to use your fingerprint, f, to "lock" the secret s. You want the locked version of your secret to have two properties:

1. You should be able to "unlock" the locked version of s
2. No one else should be able to "unlock" s

We claim that if we can solve the above problem, then we can solve the problem of designing a secure hash function for fingerprints. We leave the details as an exercise. (Hint: pick s at random and then in addition to the locked version of s also store $h(s)$, where h is an off-the-shelf secure hash function for strings.)

We will now formalize the fuzzy vault problem.

16.2.1 Formal Problem Statement

The first thing we need to do is quantize the measurements of the minutia. We cannot be infinitely precise in our measurements anyways, so let us assume that all quantized minutia, (x_i, y_i, θ_i), can be embedded into $\mathbb{F}_q$ for some large enough prime power q. Theoretically, this can also help to correct the first two issues from our naive solution. We could go through all possible values in $\mathbb{F}_q$ to get rid of translation and rotation errors (*e.g.*. for every $(\Delta x, \Delta y, \Delta z) \in \mathbb{F}_q$, we rotate and translate each minutia (x_i, y_i, z_i) to $(x_i + \Delta x, y_i + \Delta y, z_i + \Delta z)$). [1] We can also do some

[1] To be more precise we first perform the translation and rotation over the reals and then quantize and map to the appropriate $\mathbb{F}_q$ value.

local error correction to a quantized value to mitigate the effect of varying pressure. We stress that going over all possible shifts is not a practical solution, but theoretically this can still lead to a polynomial-time solution.

We now formally define our problem, which primarily captures issues 3 and 4. (Below for any integers $t \geq 1$, $\binom{\mathbb{F}_q}{t}$ denotes the set of all subsets of $\mathbb{F}_q$ of size exactly t.) The following are the components of the problem:

- Integers $k \geq 1$, $n \geq t \geq 1$
- Secret $s \in \mathbb{F}_q^k$
- Fingerprint $f \in \binom{\mathbb{F}_q}{t}$
- $\text{LOCK} : \mathbb{F}_q^k \times \binom{\mathbb{F}_q}{t} \to \binom{\mathbb{F}_q}{n}$
- $\text{UNLOCK} : \binom{\mathbb{F}_q}{t} \times \binom{\mathbb{F}_q}{n} \to \mathbb{F}_q^k$

The goal of the problem is to define the functions LOCK and UNLOCK such that they satisfy these two properties (for some $c < t$):

1. (c-completeness.) For any $f, f' \in \binom{\mathbb{F}_q}{t}$ such that $|f - f'| \leq c$, the following should hold:

$$\text{UNLOCK}\left(\text{LOCK}(s, f), f'\right) = s.$$

2. (Soundness.) It should be "hard" for an adversary to get s from $\text{LOCK}(s, f)$. (The notion of "hard" will be more formally defined later.)

Note that the completeness property corresponds to the matching property we need from our hash function, while the soundness property corresponds to the security property of the hash function.

16.2.2 Two Futile Attempts

We begin with two attempts at designing the LOCK and UNLOCK functions, which will not work. However, later we will see how we can combine both to get our final solution.

For this section, unless mentioned otherwise, we will assume that the original fingerprint f is given by the set $\{\alpha_1, \ldots, \alpha_t\}$.

Attempt 1. We begin with a scheme that focuses on the soundness property. A very simple idea, which is what we will start off with, would be to just add $n-t$ random values to f to get our vault. The intuition, which can be made precise, is that an adversary just looking at the vault will just see random points and will not be able to recover f from the random set of points. The catch of course that this scheme has terrible completeness. In particular, if we get a match between a value in the second fingerprint f' and the vault, we have no way to know whether the match is to one of the original values in f or if the match is with one of the random "chaff" points there were added earlier.

Attempt 2. Next, we specify a scheme that has good completeness (but has pretty bad soundness).

We begin with the LOCK function:

$$\text{LOCK}_2(s, f) = \{(\alpha_1, P_s(\alpha_1)), \ldots, (\alpha_t, P_s(\alpha_t))\},$$

where $P_s(X) = \sum_{i=0}^{k-1} s.X^i$ and recall $f = \{\alpha_1, \ldots, \alpha_t\}$. (Note that we have $n = t$.)

The main intuition behind LOCK_2 is the following. Given another fingerprint $f' = \{\beta_1, \ldots, \beta_t\}$ such that it is close enough to f, i.e. $|f \setminus f'| \leq c$, for every value in $f \cap f'$, we will know the corresponding P_s value and thus, we can use the fact that we can decode Reed-Solomon codes from erasures to recover the secret s. We formally present UNLOCK_2 as Algorithm 24.

Algorithm 24 UNLOCK_2

INPUT: Vault $\{(\alpha_1, y_1), \ldots, (\alpha_t, y_t)\} = \text{LOCK}(s, f)$ and another fingerprint $f' = \{\beta_1, \ldots, \beta_t\}$
OUTPUT: s if $|f \setminus f'| \leq c$

1: FOR $i = 1, \ldots, t$ DO
2: IF there exists a $j \in [t]$ such that $\alpha_i = \beta_j$ THEN
3: $z_j \leftarrow y_i$
4: ELSE
5: $z_j \leftarrow ?$
6: $\mathbf{z} \leftarrow (z_1, \ldots, z_t)$
7: Run Algorithm from Theorem 11.2.1 to correct $\mathbf{z}$ from erasures for RS codes with evaluation points $\{\beta_1, \ldots, \beta_t\}$ and output resulting message as s.

The following result is fairly simple to argue.

Lemma 16.2.1. *The pair* $(\text{LOCK}_2, \text{UNLOCK}_2)$ *of functions is* $(t-k)$*-complete. Further, both functions can be implemented in polynomial time.*

Proof. Let us assume $|f \setminus f'| \leq t-k$. Now as both f and f' have exactly t values, this means that $\mathbf{z}$ has at most $t-k$ erasures. Thus, by Theorem 11.2.1, Step 6 will output s and UNLOCK_2 can be implemented in polynomial time. Further, the claim on the polynomial run time of LOCK_2 follows from the fact that one can do encoding of Reed-Solomon code in polynomial time. □

Unfortunately, ($\text{LOCK}_2, \text{UNLOCK}_2$) pair has terrible soundness. This is because the vault $\{(\alpha_1, y_1), \ldots, (\alpha_t, \ldots$ has f in the first component in each pair. This an adversary can just read off those values and present $f' = \{\alpha_1, \ldots, \alpha_t\}$, which would imply that $\text{UNLOCK}_2(\text{LOCK}_2(s, f), f') = s$, which means that the vault would be "broken."

16.3 The Final Fuzzy Vault

So far we have seen two attempts: one that (intuitively) has very good soundness but no completeness and another which has good completeness but terrible soundness. It is natural to consider if we can combine both of these attempts and get the best of both worlds. Indeed, it turns we can easily combine both of our previous attempts to get the final fuzzy vault.

Algorithm 25 presents the new LOCK_3 function.

Algorithm 25 LOCK_3

INPUT: Fingerprint $f = \{\alpha_1, \ldots, \alpha_t\}$ and secret $s = (s_0, \ldots, s_{k-1}) \in \mathbb{F}_q^k$
OUTPUT: Vault with f locking s

1: $R, T \leftarrow \emptyset$
2: $P_s(X) \leftarrow \sum_{i=0}^{k-1} s_i \cdot X^i$
3: FOR $i = 1, \ldots, t$ DO
4: $\quad T \leftarrow T \cup \{\alpha_i\}$
5: FOR $i = t+1, \ldots, n$ DO
6: $\quad \alpha_i$ be a random element from $\mathbb{F}_q \setminus T$
7: $\quad T \leftarrow T \cup \{\alpha_i\}$
8: FOR every $\alpha \in T$ DO
9: $\quad \gamma$ be a random element from $\mathbb{F}_q \setminus P_s(\alpha)$
10: $\quad R \leftarrow R \cup \{(\alpha, \gamma)\}$
11: Randomly permute R
12: RETURN R

Algorithm 26 presents the new UNLOCK_3 function.

The following result is a simple generalization of Lemma 16.2.1.

Lemma 16.3.1. *The pair* ($\text{LOCK}_3, \text{UNLOCK}_3$) *of functions is* $(t-k)/2$*-complete. Further, both functions can be implemented in polynomial time.*

Proof. Let us assume $|f \setminus f'| \leq (t-k)/2$. Now as both f and f' have exactly t values, it implies that $|f \cap f'| \geq (t+k)/2$. Further for each $j \in [t]$ such that $\beta_j \in f \cap f'$, we have that $z_j = P_s(\beta_j)$. In other words, this means that $\mathbf{z}$ has at most $(t-k)/2$ errors.[2] Thus, by Theorem 11.2.2, Step 6 will output s and UNLOCK_3 can be implemented in polynomial time. Further, the claim on the polynomial run time of LOCK_3 follows from the fact that one can do encoding of Reed-Solomon code in polynomial time. □

[2] To be more precise if $\mathbf{z}$ has e errors and s erasures w.r.t. the codeword corresponding to s, then $2e + s \leq t - k$.

Algorithm 26 UNLOCK$_2$

INPUT: Vault $\{(\alpha_1, y_1), \ldots, (\alpha_n, y_n)\} = \text{LOCK}(s, f)$ and another fingerprint $f' = \{\beta_1, \ldots, \beta_t\}$
OUTPUT: s if $|f \setminus f'| \leq c$

1: FOR $i = 1, \ldots, t$ DO
2: IF there exists a $j \in [n]$ such that $\alpha_i = \beta_j$ THEN
3: $z_j \leftarrow y_i$
4: ELSE
5: $z_j \leftarrow ?$
6: $\mathbf{z} \leftarrow (z_1, \ldots, z_t)$
7: Run Algorithm from Theorem 11.2.2 to correct $\mathbf{z}$ from errors and erasures for RS codes with evaluation points $\{\beta_1, \ldots, \beta_t\}$ and output resulting message as s.

16.3.1 Soundness

To avoid getting into too much technical details, we will present a high level argument for why the proposed fuzzy vault scheme has good soundness. Given a vault $\{(\alpha_1, y_1), \ldots, (\alpha_n, y_n)\} = \text{LOCK}_3(s, f)$, we know that there are exactly t values (i.e. those $\alpha_j \in f$) such that the polynomial $P_s(X)$ agrees with the vault on exactly those t points. Thus, an intuitive way to argue the soundness of the vault would be to argue that there exists a lot other secrets $s' \in \mathbb{F}_q^k$ such that $P_{s'}(X)$ also agrees with the vault in exactly t positions. (One can formalize this intuition and argue that the vault satisfies a more formal definition of soundness but we will skip those details.)

We will formalize the above argument by proving a slightly different result (and we will leave the final proof as an exercise).

Lemma 16.3.2. *Let $V = \{(x_1, y_1), \ldots, (x_n, y_n)$ be n independent random points from $\mathbb{F}_q \times \mathbb{F}_q$. Then, in expectation, there are at least $\frac{1}{3} \cdot q^k \cdot \left(\frac{n}{qt}\right)^t$ polynomials $P(X)$ of degree at most $k-1$ such that for exactly t values of $j \in [n]$, we have $P(x_j) = y_j$.*

Proof. Consider a fixed polynomial $P(X)$ and a $j \in [n]$. Then for any $x_j \in \mathbb{F}_q$, the probability that for a random $y_j \in F_q$, $P(x_j) = y_j$ is exactly $1/q$. Further, these probabilities are all independent. This implies that the probability that $P(X)$ agrees with V in exactly t positions is given by

$$\binom{n}{t} \cdot \left(\frac{1}{q}\right)^t \cdot \left(1 - \frac{1}{q}\right)^{n-t} \geq \frac{1}{3}\left(\frac{n}{qt}\right)^t.$$

Since there are q^k such polynomials, the claimed result follows. □

We note that there are two aspects of the above lemma that are not satisfactory. (i) The result above is for a vault V with completely random points whereas we would like to prove a similar result but with $V = \text{LOCK}_3(s, f)$ and (ii) Instead of a bound in expectation, we would like to prove a similar exponential lower bound but with high probability. We leave the proof that these can be done as an exercise. (Hint: Use the "Inverse Markov Inequality.")

16.4 Bibliographic Notes

The fuzzy vault presented in this chapter is due to Juels and Sudan [33]. The “inverse Markov inequality" first appeared in Dumer et al. [8].