Say it ain't so - an implementation of deniable encryption

Ari Trachtenberg
trachten@bu.edu

Abstract

We are interested in the ability to lie convincingly about the contents of an encrypted file. For example, a businessman traveling through dangerous territory with sensitive documents may want, if kidnapped, to convincingly lie to his captors about the contents of the documents. Our approach encrypts a plaintext into a ciphertext plus a private key. The ciphertext can be decrypted back into the plaintext using this key, or it can be decrypted into other (decoy) plaintexts using decoy keys. The decoy plaintexts can be chosen at encryption time, or, in some cases, after encryption. Our implementation is based upon an existing framework for reconciling similar strings using little communication.

I. Introduction

This work considers the technical ability to plausibly deny or obscure the existence of certain data (as distinguished from the corresponding *political* variant). This ability is important whenever a person may be coerced to decrypt an encrypted document in a hostile environment, or when a decryption occurs under observation.

More precisely, this paper provides a means of encrypting a text into a ciphertext c that can be decrypted back into either the original text or to one of several variant texts, depending on the decryption key presented. For example, a user might encrypt the text "Don't try to meet Angelina" using a key k into a ciphertext c, which could then be decryptable back to the original text with key k, but to a different text, say "Don't cry for me Argentina", when decrypted with a different key k', and possibly yet another text using an even different key k''.

Consider three example scenarios to motivate this need.

a) Rubberhose: In the most straightforward scenario, a businessman is traveling through dangerous territory with sensitive documents. If kidnapped with his laptop, the businessman may be coerced (e.g. with a traditional rubberhose) to decrypt its contents or face emotional or physical consequences. Such a person might satisfy his kidnappers by decrypting the contents with a decoy key that produces plausible but modified (or redacted) plain text.

b) Eavesdropping: Another scenario involves a woman who feels she is being observed (e.g. by an abusive spouse) during communications with third parties. She could (incorrectly) alleviate the suspicion of her observer by decrypting sensitive communication with a decoy key (or by releasing the decoy key to observation) without compromising the correspondence.

c) Voting: A final scenario involves voting in an untrusted environment, where, for example, communication between the voting machine and the national authority is being observed. A voter may vote using a secret shared with the national authority, and, if coerced to reveal her vote locally based on eavesdropped communication, may decrypt the communication to suggest a different vote.

The author is with the department of electrical and computer engineering at Boston University. Part of this work was completed when the author was a visiting professor at the Technion Computer Engineering center in Haifa, Israel.

Black Hat Asia 2014, Singapore

II. Literature

A great body of work exists in the general area of technical plausible deniability.

The simplest method of providing deniable security involves the one-time pad, wherein a user XORs his plaintext with an equally long collection of random bits (i.e. his key) to produce a ciphertext. By using a differing sequence of random bits, the user can decode the ciphertext into any string of equal (or lesser) length than his plaintext. This approach provides *information theoretic security*, in that ciphertext alone provides no information about the content of the string, but it suffers from an exceedingly long key.

This approach and its more sophisticated analogs are known in the cryptography community under the name of deniable encryption [5]. In the general framework, the user employs random bits r in his encryption of a message m; in the face of coercion, the user produces a different message m' and random bits r' that also produce the same encryption, leading the coercer to believe that this was the original message. A modification in [9] based on ElGamal encryption and an established secret allows a user to encrypt one text and decrypt to one of two texts, and there is a fair amount of other related work (e.g. [13]) in the literature.

Forms of deniable encryption have also been implemented in code, with notable examples such as the Rubberhose File System [1] (no longer supported), StegFS [11], [4] and TrueCrypt [2]. In its "hidden volume" usage, the latter hides data steganographically in random-appearing bits on the free space of a volume. Since TrueCrypt always writes random data into free space, an adversary will not know whether the free space is truly free or contains hidden data.

In contrast with these approaches, we are interested in the ability to deterministically encrypt a text with one key in such a way that it can be decrypted either to the original text, or to a variety of other texts, depending on the decryption key provided by the user. Our approach is information-theoretic, rather than computational, in nature, in that the adversary cannot distinguish between possible decryptions without side-information. In our case, the alternate plaintexts can be chosen up front in a "plan-ahead" fashion, or, in some cases, after encryption has been completed.

III. Formulation

Formally, we wish to develop two functions:

- an *encoding* function
$$\mathrm{encrypt}(k_0, T_0) \longrightarrow C$$
that encodes plaintext T_0 with an authentic key k_0 into a ciphertext C, and
- a *decoding* function
$$\mathrm{decrypt}(k, C)$$
that produces plaintext from the given ciphertext C with a number (≥ 1) of possible keys k, although decoding with k_0 produces the plain text (i.e. $\mathrm{decrypt}(k_0, C) = T_0$).

The key k_0 is the *authentic* key for the ciphertext, and all other keys that produce text are *decoy* keys. It may be desirable to engineer the encoding/decoding so that certain decoy keys will produce predetermined alternate texts. For example, different decoding keys might produce texts that are subtly different from the original text (but in important details).

IV. Background

Our approach is based on a solution to the *set reconciliation* problem [12], which we briefly describe hereafter, together with one of its applicable variants. A different approach to this reconciliation is described in [6] with reference to fuzzy extractors.

A. Set reconciliation

In this problem, two remote hosts A and B have, respectively, similar sets S_A and S_B of b-bit vectors; the approach we describe can also be adapted for multi-sets, which allow multiple instances of the same element.

The hosts seek to determine the mutual difference of their sets

$$S_A \oplus S_B = (S_A - S_B) \cup (S_B - S_A)$$

using as little communication as possible. As a simple example, if one host has $S_A = \{1, 3, 4, 5\}$ and the other host has $S_B = \{1, 3, 5, 6, 7\}$, then the goal would be to convey information so that both hosts learn that A is missing 6, and 7, while B is missing 4. We will use the letter m to denote the number of differences between the two sets (in this case, 3).

The solution proposed in [12] involves the use of a characteristic polynomial to mathematically determine the differences between the two sets. We briefly sketch the algorithm below, which is implemented over arithmetic in a finite field, but we refer the reader to the paper for a more complete explanation and examples:

- Host A generates the polynomial $\chi_{S_A}(x) = \prod_{s \in S_A}(x - s)$, and similarly Host B generates $\chi_{S_B}(x)$.
- Host A evaluates $\chi_{S_A}(x)$ on m points and exchanges these with similar evaluations on host B.
- Hosts A and B interpolate the unique rational function $f(x)$ that matches $\frac{\chi_{S_A}(x)}{\chi_{S_A}(x)}$ on the m evaluation points.
- Hosts A and B each factor $f(x)$ to get exactly those elements in the mutual difference of sets S_A and S_B.

The key feature of this reconciliation algorithm is that its communication is linearly tied to the number of differences between the two sets, and only logarithmically tied to the sizes of the sets themselves. Thus, if two sets of one billion elements differ in just 5 elements, then communication of roughly five 20-bit packets is enough to determine these elements.

B. String reconciliation

This reconciliation approach can be extended to remote strings and not just sets, the main difference being that the order of set elements is not significant within a set but the order of string characters is significant [3]; for example, sets $1, 2, 3$ and $3, 2, 1$ are identical, whereas strings `123` and `321` are not. The general idea then is to start by *shingling* each string into contiguous, overlapping substrings, so that the string `Hello_There` is transformed into the multiset of length 4 shingles:

$$S = \{\texttt{Hell, ello, llo\_, lo\_T, o\_Th, \_The, Ther, here}\}.$$

This multiset of shingles can always be put back together into the original string by combining overlapping shingles in the same order as they appear in the original string. However, there may be cases where the shingles can be consistently combined into a different string [10]; for example, the string `katana` has the following shingling into length 2 substrings:

$$S' = \{\texttt{ka,at,ta,an,na}\}.$$

However, S' can also be put together into the string `kanata`.

The work in [7] provides an efficient algorithm for determining when a given collection of shingles can be uniquely decoded into a string, together with an algorithm for merging nearby shingles, if necessary, to provide this unique decodability property.

V. The approach

We next describe our approach for meeting the goals of our deniable encryption formulation from Section III.

A. *Encryption* - encrypt

The encryption of a plaintext p makes use of two parameters: δ, which affects the number of decoy texts that can be extracted, and ϵ, which affects the length and security of the private key. Encryption proceeds as follows:

1) Shingle text p into a multiset of substrings S_p. If S_p is not uniquely decodable, increase the length as needed to guarantee this property.
2) Pick a subset $S_i \subseteq S_p$ of size at most $|S_p| - \delta$ and generate the characteristic polynomial $\chi_{S_i}(x)$.
3) The encrypted text C is the evaluation of $\chi_{S_i}(x)$ at $|S_i| - \epsilon$ publicly known points.
4) The user's private key k_0 is the union of $S_p - S_i$ and any ϵ elements of S_i.

B. *Decryption* - decrypt

To decrypt C into the original plaintext p:

1) Compute $|S_i| - \epsilon$ evaluations of the characteristic polynomial $\chi_{k_0}(x)$ of the elements in his private key.
2) Utilize these evaluations together with those published in C to produce the multiset S_p. This is done using the set reconciliation algorithm detailed above from [12].
3) Combine the shingles in S_p (in the only way possible) to produce the plaintext p.

C. *Deception*

The user may deceive an observer into believing that he has decrypted C into an (incorrect) text p' as follows:

1) The user computes a new key k_0' as the union of $S_{p'} - S_i$ and any ϵ elements of S_i. Here we need that $S_{p'}$ is comprised of S_i and at most δ additional shingles.
2) Decoding with the new key should produce the different shingle multiset $S_{p'}$, which should decode to a different plaintext.

The user can chooses as a decoy any multiset $S_{p'} \supseteq S_i$ with the property that $|S_{p'} - S_i| \leq \delta$. As such, he can implement "plan-ahead" deniability by producing several similar versions of a text and precomputing keys that produce each one, or he can compute decoy keys after the encryption as long as they satisfy the deception requirements.

VI. Analysis

The work in [3] shows that a shingle length l roughly logarithmic in the length of p is sufficient for unique decodability of random text (*i.e.,* a string of identical, identically distributed random characters), and empirical evidence suggests that this is the case for English text as well [8]. Although it is possible to get unique decodability with shorter lengths, say by merging certain shingles [7], our analysis herein assumes a shingle length logarithmic in plaintext length for purposes of simplicity of exposition.

A. *Algorithm*

In the following work, we assume an input plaintext of length n characters.

Theorem 1 *A ciphertext produced by* encrypt *has length* $O\left((n-\epsilon)\log n\right)$ *bits and requires* $O(n^2 \log n)$ *time to compute.*

Note that the ciphertext is at most logarithmically longer the plaintext, but it can also be shorter (if ϵ is large).

Proof: The length of the encrypted plaintext is $|S_i| + \delta - \epsilon$ characteristic polynomial evaluations in a finite field of size $\log(n)$.[1] For a plaintext of length n, $|S_i| \leq (n - \delta) - l + 1$ by design, and, with logarithmically-sized shingles, this produces an overall ciphertext length that is as described. Once a shingle length is determined, the entire encryption requires at most n evaluations of a polynomial of degree at most n over a finite field whose size is exponential in shingle length, and this dominates the computation time. ∎

Theorem 2 *A key produced by* encrypt *has length* $O((\delta + \epsilon) \log n)$ *bits. For constant* δ *and* ϵ*, decoding requires time* $O(n^3)$.

The length of the key is at most $\delta + \epsilon$ characteristic polynomial evaluations (plus metadata), or roughly $(\delta + \epsilon) \log n$ bits for logarithmically sized shingles. When δ and ϵ are small, this is logarithmic in the plaintext size; in this case, decoding complexity is dictated by the cost of set reconciliation. Altogether, the ciphertext and key represent roughly a logarithmic overhead over the information-theoretic content of the average plaintext (i.e. n characters).

B. Security

The security guarantees are summarized in the following theorem.

Theorem 3 *No shingle in a ciphertext can be decoded with certainty without knowledge of* $\epsilon \log(n)$ *additional bits.*

This follows because the ciphertext represents $|S_i| - \epsilon$ equations of a linear system with $|S_i|$ equations that uniquely determine a part of the plaintext.

Plan-ahead encryption: Any collection of plaintexts $T_j, j = 1 \ldots k$ can be encrypted into one ciphertext C and keys $k_j, j = 1 \ldots k$. The length of the ciphertext will be determined by the number of shingles common to the plaintexts.

To accomplish this encryption, the user finds the intersection of the shingles of all texts, and decrees this to be S_i in the encryption algorithm *ENC* (shingles can be randomly deleted to increase security). The key for text T_j will thus be the difference between the shingles for T_j and S_i.

After-the-fact encryption: After encryption, keys can also be added for some additional texts. The following theorem provides the minimum number of additional texts that can be supported (although, in practice, the number should be much higher).

Theorem 4 *For long* n*, there are at least* $\frac{\delta}{\log n}$ *decoy texts that be decoded from any given ciphertext produced by* ENC.

Proof: One may add at least δ shingles to S_i to produce a decryptable decoy text. Since each insertion of a character into the text adds at most $\log n$ shingles, the result follows. ∎

Finally, we note that knowledge of a specific decoy key does not provide any specific information about the other decoy keys (beyond that they are not the same); it does, however, provide information about S_i, which contains some of the shingles of the plaintext.

VII. Considerations

We next detail several natural considerations of our approach.

[1]It is possible that S_i does not contain ϵ elements (i.e. it is too small), in which case encryption fails with a suggestion that the security parameter cannot be supported.

A. Hashed keys

In the current implementation, keys are comprised of plaintext shingles, which makes them vulnerable to dictionary attacks. One solution is to encrypt the key using a one-time pad and include this encryption in the ciphertext; the random characters added to form the one-time pad become the new key. For example, if the key $k = \{\texttt{TES, EST}\}$, then it can be character-wise added to a random $r = \{\texttt{NES, OTH}\}$ to produce the cipher $\{\texttt{GIK,SLA}\}$, which is appended to the ciphertext, and the new key becomes r.

B. Shortened keys

Several techniques exist for shortening keys.

- *Shingle compression* - Key shingles that overlap can be combined into one longer text from which the shingles can be extracted upon decoding. For example, a shingle `[ b l a g i ]` can be combined with the shingle `[ l a g i s ]` to produce `[ b l a g i s ]`.
- *Flexible shingle length* - Rather than increasing all shingle lengths until a uniquely decodable string is produced, only a few specific shingles can be merged [7], leaving the remaining shingles to be short.
- *Re-encoding* - Where decoy keys have similarity, they themselves may be encoded using $\mathrm{encrypt}$.

C. Large files

For large files, computational complexity is a serious bottleneck for the current implementation. Possible solutions include block-based encryption and using more efficient set reconciliation algorithms at the expense of larger ciphertexts.

ACKNOWLEDGMENTS

The author would like to thank Leo Reyzin for suggesting hiding key information using a one-time as described in Section VII and for providing thoughtful feedback on an earlier draft.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1012910. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Robberhose (file system). No longer supported.
[2] Truecrypt: Free open-source on-the-fly encryption. http://truecrypt.org.
[3] Sachin Agarwal, Vikas Chauhan, and Ari Trachtenberg. Bandwidth efficient string reconciliation using puzzles. *Parallel and Distributed Systems, IEEE Transactions on*, 17(11):1217–1225, 2006.
[4] Ross Anderson, Roger Needham, and Adi Shamir. The steganographic file system. In *Information Hiding*, pages 73–82. Springer, 1998.
[5] Rein Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In *Advances in Cryptology-CRYPTO'97*, pages 90–104. Springer, 1997.
[6] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.
[7] Arnold Filtser, Jiaxi Jin, Aryeh Kontorovich, and Ari Trachtenberg. Efficient determination of the unique decodability of a string. In *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, pages 1411–1415. IEEE, 2013.
[8] J. Jin. Prioritized data synchronization with applications. Master's thesis, Boston University, 2012.
[9] Marek Klonowski, Przemysaw Kubiak, and Mirosaw Kutyowski. Practical deniable encryption. In Viliam Geffert, Juhani Karhumki, Alberto Bertoni, Bart Preneel, Pavol Nvrat, and Mria Bielikov, editors, *SOFSEM 2008: Theory and Practice of Computer Science*, volume 4910 of *Lecture Notes in Computer Science*, pages 599–609. Springer Berlin Heidelberg, 2008.
[10] Aryeh Kontorovich and Ari Trachtenberg. Deciding unique decodability of bigram counts via finite automata. *Journal of Computer and System Sciences*, 2013.
[11] AndrewD. McDonald and MarkusG. Kuhn. Stegfs: A steganographic file system for linux. In Andreas Pfitzmann, editor, *Information Hiding*, volume 1768 of *Lecture Notes in Computer Science*, pages 463–477. Springer Berlin Heidelberg, 2000.
[12] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Trans. on Info. Theory*, September 2003.
[13] Adam O'Neill, Chris Peikert, and Brent Waters. Bi-deniable public-key encryption. Cryptology ePrint Archive, Report 2011/352, 2011. http://eprint.iacr.org/.