Data Storage By Secure Crumbling With Signing Trusted Third Parties

CYRIL DEVER

Edgewhere

September 18, 2020

Abstract

We define a secure data storage solution based on the presence of one (or more) trusted third parties necessary to perform encryption and decryption operations on a message split in crumbs. This secure storage method is particularly safe since the encryption elements are distributed among the different participants and can't be discovered by a single procedure which would allow breaking a unique encryption code. We show that this distribution of crumbs and their separate encryption considerably increases the security of the storage since, in the absence of a participant, the message can't be recovered. Furthermore, the algorithm doesn't allow anyone other than the rightful owner of the original message to know in clear all or part of the data at any time whatsoever. This technique is pending patent.[*]

I. INTRODUCTION

There are already multiple available ways to store data after encrypting it. However, the current techniques of data encryption for the storage and recovery of stored data and their decryption are operations all the more complex as the security must be high.

This complexity comes with the added burden of the risk that the encryption key is always susceptible to being broken and/or hacked.

The goal of our new algorithm, called the `crumbl`® technology, is to develop simple yet particularly effective means for securing data storage.

Our procedure describes a method of secure storage of a source data, owned by one (or more) *holder(s)*, using already proven techniques of asymmetric encryption with the participation of so-called trusted third parties, each having a pair of private and public keys.

II. BASIC DEFINITIONS

Definition 1 (Source Data). The source data d is the data that has to be protected by the `crumbl` encryption protocol.

Definition 2 (Crumbl). A *crumbl* (or crumbled string) is the final result of the encryption

of a source data through the `crumbl` process. Among other elements, it uses crumbs which come from slices of the source data.

Definition 3 (Crumb). A crumb ς is an encrypted portion of data of size n in its binary form:

$$\varsigma := \sum_{j=0}^{n-1} x_j \mid x_j \in \{0,1\} \qquad (1)$$

It could be the byte array itself or any string representation of it (hexadecimal, binary, base-64, . . .).

When presented with a lower index (eg. ς_8), it indicates the order (starting at 0) in which to eventually concatenate it with the others. With an added upper index (eg. ς^π), it indicates its signer (π) during encryption.

A set of crumbs can only be assigned to one source data. In other words, it is obvious that one can't mix a crumb $c1$ from a data $d1$ with a crumb $c2$ from a data $d2$.

Definition 4 (Slice). A slice σ is a padded plaintext portion of the source data.

Let $\mu()$ be a padding function and $\mu^{-1}()$ its inverse. For t slices made out of a source data d, we have:

$$\begin{cases} \sigma_i := \mu\left[\left(\frac{d}{t}\right)_i \right] \\ \\ d := \mu^{-1}(\sigma_0) \parallel \mu^{-1}(\sigma_1) \parallel \ldots \parallel \mu^{-1}(\sigma_{t-1}) \end{cases} \qquad (2)$$

III. The Protocol

Definition 5 (Operation). An operation takes a source data and encrypts it with the `crumbl`, or back.

Definition 6 (Participant). A participant $\pi \in P$ (or signer) is defined by his pair of public (PK) and private (SK) keys unique to an `crumbl` operation he is taking part along with other participants/signers.

$$\begin{aligned} \pi : P &\rightarrow (\mathcal{K} \times \mathcal{K}) \\ \pi_i &\mapsto (\pi_i^{SK}, \pi_i^{PK}) \end{aligned} \qquad (3)$$

There are two kinds of participants involved in the process:

- The holders who wish to protect their asset, ie. the source data;
- The trusted third parties, generally being corporations and the main sponsors of the system, who only participate in data encryption/decryption as signers and are paid for it.

Definition 7 (Holder). The holder is the only participant able to have access to the data in clear, ie. the source data. He could be the rightful owner of the data or anyone to whom the latter delegates its use.

He is (or they are, should there be more than one holder involved in an operation) the signer(s) of a special crumb: ς_0, ie. the one with index 0.

There must be at least one holder and one trusted third party in the list of participants[1].

1. Encryption

Algorithm 1 presents the encryption protocol of the `crumbl`® technology.

Let p be the number of participants forming the set $P \leftarrow \{\pi_p\}$ of signers, $P_0 \in P$ the subset of holders, and $P_\tau \in P$ the subset of trusted third parties with $P_0 \cup P_\tau = P$.

And let H be the holder of the source data d.

Finally, let $\mathfrak{c}()$ be the encryption function of the `crumbl`[2]:

$$\begin{aligned} \mathfrak{c} : \quad \omega \times \mathcal{K} &\rightarrow \omega \\ (msg, pubkey) &\mapsto \mathfrak{c}(msg, pubkey) \end{aligned} \qquad (4)$$

Algorithm 1: Encryption protocol

Input: d, P
Output: Cr

1 **if** $|d| = 0 \vee |P| < 2$ **then**
2   **throw** *error*

3 initialize a new set of crumbs: $\mathcal{C} \leftarrow \varnothing$;
4 ask all participants $\pi_i \in P \setminus \pi_H$ for their new public key;
5 each participant π_i creates a new pair of keys along with a request ID π_i^{RID}, this tuple being stored for future use in the decryption process;
6 d is prepared and split into a set of t slices $\{\sigma_0, \ldots, \sigma_{t-1}\}$ with: $t = |P_\tau| + 1$;
7 H encrypts σ_0 with his own new public key:

$$\mathcal{C} \leftarrow \varsigma_0^{\pi_H} := \mathfrak{c}_{\pi_H}(\sigma_0, \pi_H^{PK})$$

8 **while** *H receives each participant's public key (π_i^{PK})* **do**
9   **if** $\pi_i \in P_0$ **then**
10   H encrypts σ_0:

$$\mathcal{C} \leftarrow \varsigma_0^{\pi_i} := \mathfrak{c}_{\pi_i}(\sigma_0, \pi_i^{PK})$$

11   **else**
12   all other slices are encrypted by H with the received public key:

$$\begin{aligned} \forall j \in \{\sigma_1, \ldots, \sigma_{t-1}\} : \\ \mathcal{C} \leftarrow \varsigma_j^{\pi_i} := \mathfrak{c}_{\pi_i}(\sigma_j, \pi_i^{PK}) \end{aligned}$$

13 Cr is finalized by H using d and the set of all crumbs $\mathcal{C}$;
14 **return** Cr

As shown, everything takes place in H envi-

---

[1] We shall see that maximum security starts with at least four participants: one holder and three trusted third parties.

[2] It could be any asymmetric protocol as long as it is available for H. Thus, we assume that H knows beforehand which protocol uses each participant π_i; therefore, he'd be using the appropriate $\mathfrak{c}_{\pi_i}()$ function.

ronment which guarantees that the source data is never sent, let alone known, by any other stakeholder.

If no error is raised, the output crumbl *Cr* can be stored anywhere, by any of the stakeholders and/or some outsourcer (eg. a hosting service). In any case, *H* should store a tuple of references to *Cr* (or *d*) and the keypair used for the operation. He may also store its verification hash and use it for that purpose.

Definition 8 (Verification Hash). A verification hash *V* is made of the concatenation of the 32 first characters of a crumbled string *Cr*:

$$V := \|_{i=1}^{32} Cr[i] \tag{5}$$

where $Cr[i]$ is the *i*-th character of *Cr*.

The verification hash is unique to an operation — see Definition 9.

It is generally used for search or storage purposes. For example, our latest implementation requires that we ask a hosting service for a crumbl by sending its verification hash.

By definition, it is verified that[3]: $V \subset Cr$.

2. DECRYPTION

Algorithm 2 presents the decryption protocol.

Let P'_τ a subset of P_τ of size $1 \le n \le |P_\tau| - 1$, and H_1 one of the signing holders that wishes to recover the data.

And let $HR()$ be the hashered function — see (7).

Finally, let $\mathfrak{D}()$ be the decryption function of the `crumbl`:

$$\mathfrak{D}: \quad \mathbb{N} \times \omega \times \mathcal{K} \quad \to \omega \atop (j, Cr, privkey) \quad \mapsto \mathfrak{D}(j, Cr, privkey) \tag{6}$$

with *j* being the *j*-th crumb in *Cr*.

Lorem ipsum ...

Definition 9 (Hashered Prefix). Lorem ipsum ...

$$HR: \tag{7}$$

---

[3]We will use this notation in the rest of the document when we want to assert that a passed *V* is *Cr*'s appropriate verification hash.

Algorithm 2: Decryption protocol

Input: *Cr*, a decrypter $\pi_{H_1} \in P_0$, P'_τ, an optional *V*

Output: *d*

1 **if** $V \ne \varnothing \wedge V \not\subset Cr$ **then**
2   **throw** *error*

3 **else if** $V = \varnothing$ **then**
4     $V \leftarrow \|_{i=1}^{32}(Cr)$;
5 initialize a new set of slices $S \leftarrow \varnothing$;
6 from *Cr*, we set the number of needed slices *t*;
7 **for** $i \leftarrow 0$ **to** $p = |P'_\tau| - 1$ **by** 1 **do**
8   *H1* requests his decrypted crumbs to trusted third party π_i;

9 **foreach** *participant* π_i **do**
10   initialize $j \leftarrow 1$;
11   **repeat**
12   **if** $\exists \sigma_j := \mathfrak{D}(j, Cr, \pi_i^{SK})$ **then**
13   π_i sends his slice σ_j;
14       $j \leftarrow j + 1$;
15   **until** $j = t$;

16 **while** $|S| \ne t$ AND
17   *H1 receives new slice* σ_j **do**
18   **if** $\sigma_j \notin S$ **then**
19       $S \leftarrow \sigma_j$;

20 *H1* decrypts his own crumb: $S \leftarrow \mathfrak{D}(0, Cr, \pi_{H1}^{SK})$;
21 use (2) to recover *d*:

$$d \leftarrow \mu^{-1}(\sigma_0) \| \mu^{-1}(\sigma_1) \| \ldots \| \mu^{-1}(\sigma_{t-1})$$

22 **if** $HR(d) \ne V$ **then**
23   **throw** *error*

24 **return** *d*