KryptonIT: The Discovery of a New Cryptographic System for Storing Secrets

Reto E. Koenig

University of Fribourg
&
Bern University of Applied Sciences

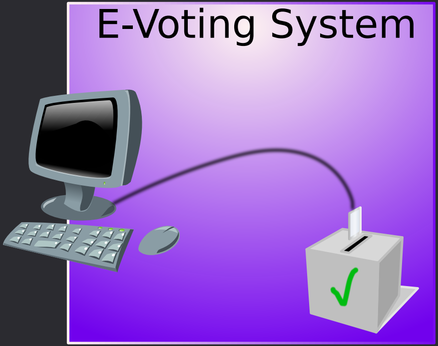
25.03.2011

UNIVERSITAS FRIBURGENSIS
Berner Fachhochschule
HASLERSTIFTUNG

What you should know about building new cryptographic systems

Ronald Rivest: “Never build your own cryptographic system!”

Current E-Voting Systems



Voter's view on a current E-Voting Systems

- It is Understandable
- It is Simple to Use
- Fast
- "Cheap"

Adversary's view on a current E-Voting Systems
Do not Underestimate the Power of the Coercer

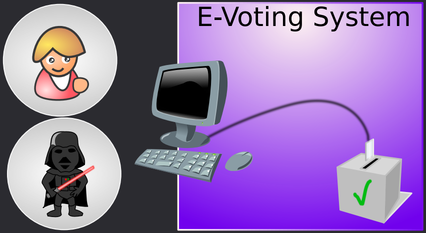
- You Shall Not Vote
- You Shall Vote
- You Shall Vote As I Say
- I randomize Your Vote
- I Vote for You

- I am Watching You
- I Know How You Voted Last Summer

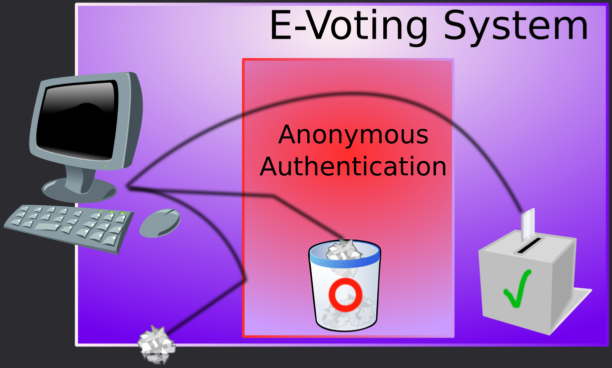
- It is Fast
- It is Scalable
- It is "Cheap"

Overall view on a current E-Voting Systems

- It is Simple
- It is Fast
- It is Cheap
- It is completely insecure



Cryptographer's new Hope



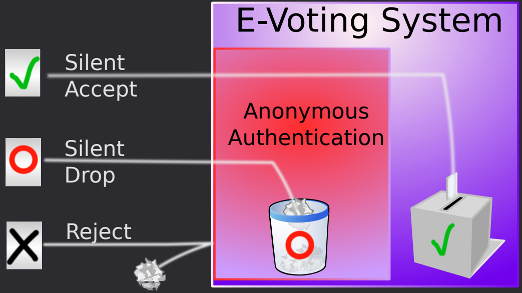
Cryptographer's view on their E-Voting Systems

- It is Coercion-Resistant
- It is End-To-End Verifiable
- # It is secure
- The voter has to remember "some" "secrets"

Some Secrets?

E2E verifiable coercion resistant E-Voting systems require:

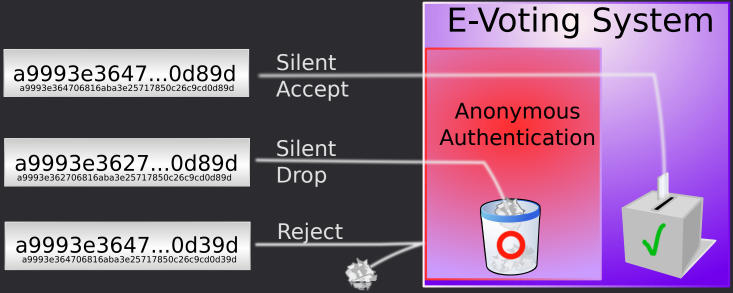
- 3 - different kind of credentials with very high entropy
- kept top secret by each voter



Some Secrets?

E2E verifiable coercion resistant E-Voting systems require:

- 3 - different kind of credentials with very high entropy
- kept top secret by each voter



A Realistic Example With 20 Credentials

In order to render the E-Voting System coercion resistant, each voter...

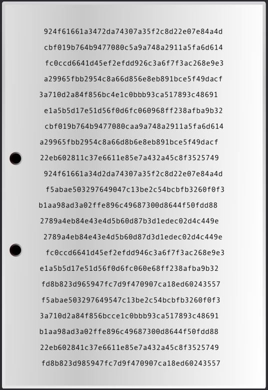
- ...needs to secretly store several dozens credentials
- ...has to discriminate doubtless between credentials for 'Accept' and 'Drop'.[a]
- ...is not allowed to mark any credential
- ...shall never unveil the amount of possessed secrets (They vary per voter)

[a] E-voting system accepts both without returning any hint

A Realistic Example With 20 Credentials

In order to render the E-Voting System coercion resistant, each voter...

- ...needs to secretly store several dozens credentials
- ...has to discriminate doubtless between credentials for 'Accept' and 'Drop'.[a]
- ...is not allowed to mark any credential
- ...shall never unveil the amount of possessed secrets (They vary per voter)



[a] E-voting system accepts both without returning any hint

A Realistic Example With 20 Credentials

In order to render the E-Voting System coercion resistant, each voter…

- …needs to secretly store several dozens credentials
- …has to discriminate doubtless between credentials for 'Accept' and 'Drop'.[a]
- …is not allowed to mark any credential
- …shall never unveil the amount of possessed secrets (They vary per voter)

[a] E-voting system accepts both without returning any hint

But How Should the Voter Store those Secrets?

Indeed it is not possible to store these secrets with some keys (passwords) in a current crypto-systems. ($\rightarrow$ Discussion)

Our View on the Secret-Storage System

The system...

- ...allows to choose freely n (usually low entropy) keys
- ...allows to choose freely n (usually high entropy) secrets
- ...has to store multiple secrets in one storage (aka cipher)
- ...has to retrieve only the secret correlated to the key
- ...has to have all properties of a (symmetric) crypto-system

Let's get Formal...

Prerequisits

$\mathcal{S}$ =*secret space*, set of all possible secrets (typically high-entropy)

$\mathcal{K}$ =*key space*, set of all possible keys (typically low-entropy)

$S = (s_1, \ldots, s_n)$, $s_i \in \mathcal{S}$, an n-tuple of (not necessarily distinct) secrets ($n \geq 1$)

$K = (k_1, \ldots, k_n)$, $k_i \in \mathcal{K}$, an n-tuple of distinct keys $n \geq 1$

$\mathcal{C}$ =*storage space*, the set of all possible storages

c =a particular storage

Functions of the Secret-Storing System

store : $\mathcal{S}^n \times \mathcal{K}^{(n)} \longrightarrow \mathcal{C}$
storage function, where $\mathcal{K}^{(n)} \subseteq \mathcal{K}^n$ is the set of all admissible key tuples (with distinct keys)

retrieve : $\mathcal{C} \times \mathcal{K} \longrightarrow \mathcal{S}$
the *retrieval function*

$\text{store}_K(S) =$ $c \in \mathcal{C}$, storing the n-tuple of the secrets $S \in \mathcal{S}^n$ with the n-tuple of distinct keys $K \in \mathcal{K}^{(n)}$

$\text{retrieve}_{k_i}(c) =$ $s_i \in \mathcal{S}$ retrieval with key k_i

---

$$\text{retrieve}_{k_i}(\text{store}_K(S)) = s_i$$

Properties of the Secret-Stroring System

Required to possess the cryptographic properties of a traditional symmetric crypto-system:

- Retrieving s_i from c does not disclose any information about the other secrets in c
- Applying K on c returns S
- Serves a conditional entropy $H(S|c)$ which is equal to $H(S)$
- Applying K' on c where $K' \neq K$ does return S with a probability of $\frac{1}{|S|}$

Definition

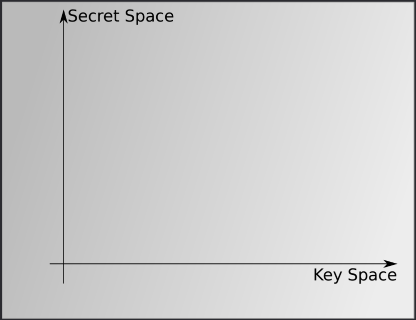
A *secret-storing system* of order n,

$$\Sigma = (\mathcal{S}, \mathcal{K}, \mathcal{C}, \text{store}, \text{retrieve}),$$

consists of a secret space $\mathcal{S}$, a key space $\mathcal{K}$, a storage space $\mathcal{C}$, and two functions store and retrieve with properties as introduced above.

An Example in $\mathbb{R}^2$ (Storing)

$$S = (s_1, \ldots, s_i, \ldots, s_n)$$
$$K = (k_1, \ldots, k_i, \ldots, k_n)$$
$$c = \text{store}_K(S)$$
$$c = \text{LagrangeInterpPol}_K(S)$$
$$c = \sum_{z=0}^{t} a_z x^z$$



An Example in $\mathbb{R}^2$ (Storing)

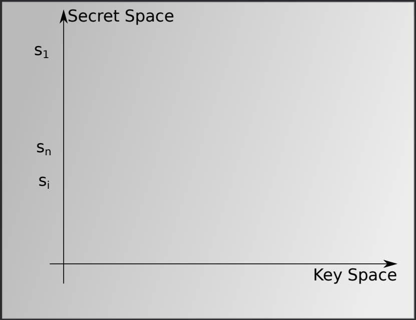
$$S = (s_1, \ldots, s_i, \ldots, s_n)$$

$$K = (k_1, \ldots, k_i, \ldots, k_n)$$

$$c = \text{store}_K(S)$$

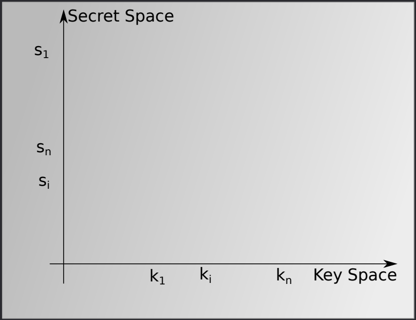
$$c = \text{LagrangeInterpPol}_K(S)$$

$$c = \sum_{z=0}^{t} a_z x^z$$



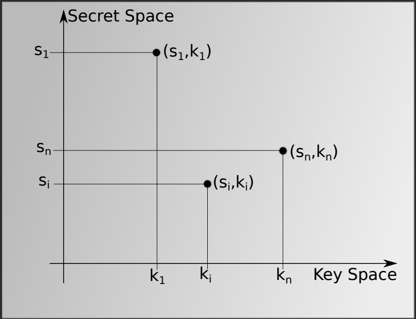
An Example in $\mathbb{R}^2$ (Storing)

$S = (s_1, \ldots, s_i, \ldots, s_n)$
$K = (k_1, \ldots, k_i, \ldots, k_n)$
$c = \text{store}_K(S)$
$c = \text{LagrangeInterpPol}_K(S)$
$c = \sum_{z=0}^{t} a_z x^z$



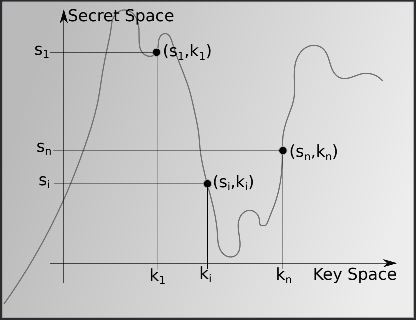
An Example in $\mathbb{R}^2$ (Storing)

$$S = (s_1, \ldots, s_i, \ldots, s_n)$$
$$K = (k_1, \ldots, k_i, \ldots, k_n)$$
$$c = \text{store}_K(S)$$
$$c = \text{LagrangeInterpPol}_K(S)$$
$$c = \sum_{z=0}^{t} a_z x^z$$



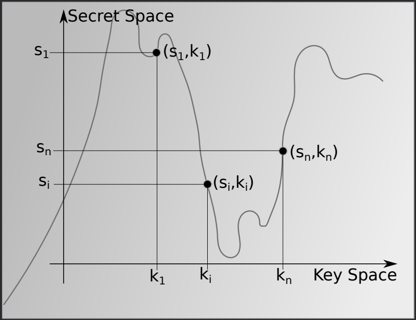
An Example in $\mathbb{R}^2$ (Storing)

$$S = (s_1, \ldots, s_i, \ldots, s_n)$$
$$K = (k_1, \ldots, k_i, \ldots, k_n)$$
$$c = \mathrm{store}_K(S)$$
$$c = \mathrm{LagrangeInterpPol}_K(S)$$
$$c = \sum_{z=0}^{t} a_z x^z$$



An Example in $\mathbb{R}^2$ (Storing)

$$S = (s_1, \ldots, s_i, \ldots, s_n)$$
$$K = (k_1, \ldots, k_i, \ldots, k_n)$$
$$c = \text{store}_K(S)$$
$$c = \text{LagrangeInterpPol}_K(S)$$
$$c = \sum_{z=0}^{t} a_z x^z$$

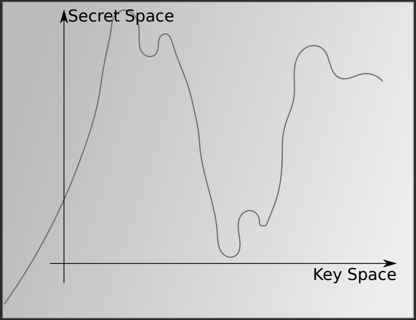


An Example in $\mathbb{R}^2$ (Retrieving)

$$c = \sum_{z=0}^{t} a_z x^z$$

$$s_i = \text{retrieve}_{k_i}(c)$$

$$s_i = f(k_i) = \sum_{z=0}^{t} a_z {k_i}^z$$

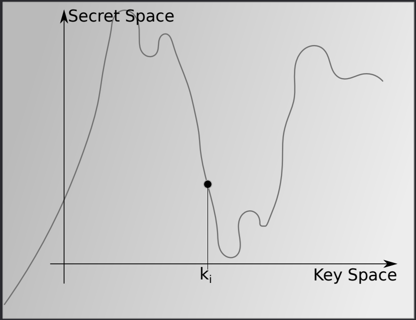


An Example in $\mathbb{R}^2$ (Retrieving)

$$c = \sum_{z=0}^{t} a_z x^z$$

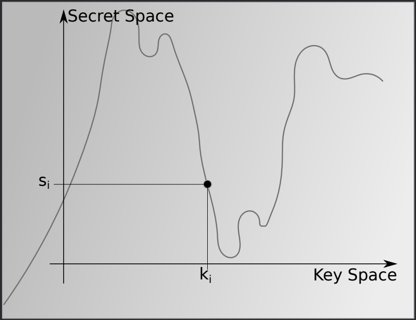
$$s_i = \text{retrieve}_{k_i}(c)$$

$$s_i = f(k_i) = \sum_{z=0}^{t} a_z {k_i}^z$$



An Example in $\mathbb{R}^2$ (Retrieving)

$$c = \sum_{z=0}^{t} a_z x^z$$
$$s_i = \text{retrieve}_{k_i}(c)$$
$$s_i = f(k_i) = \sum_{z=0}^{t} a_z {k_i}^z$$

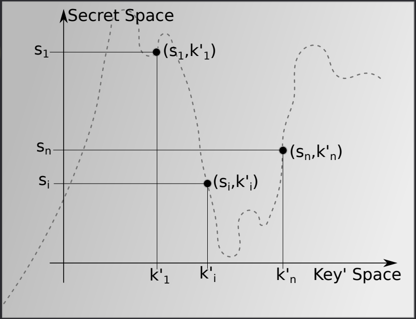


Considerations

$S = (s_1, \ldots, s_i, \ldots, s_n)$
$K = (k_1, \ldots, k_i, \ldots, k_n)$
$c = \text{store}_K(S)$
$K' = (H(k_1), \ldots, H(k_i), \ldots, H(k_n))$
$c = \text{LagrangeInterpPol}_{K'}(S)$
$c = \sum_{z=0}^{t} a_z x^z$

$s_i = \text{retrieve}_{k_i}(c)$
$k'_i = H(k_i)$

$s_i = f(k'_i) = \sum_{z=0}^{t} a_z k'^{z}_i$

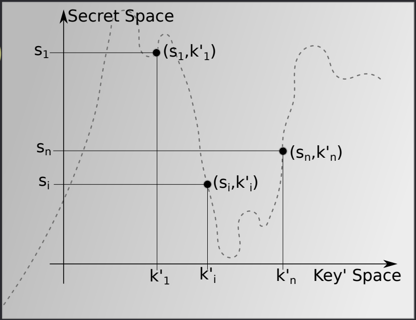


Considerations

$S = (s_1, \ldots, s_i, \ldots, s_n)$
$K = (k_1, \ldots, k_i, \ldots, k_n)$
$c = \text{store}_K(S)$
$K' = (H(k_1), \ldots, H(k_i), \ldots, H(k_n))$
$c = \text{LagrangeInterpPol}_{K'}(S)$

$c = \sum_{z=0}^{t} a_z x^z$

$s_i = \text{retrieve}_{k_i}(c)$
$k'_i = H(k_i)$
$s_i = f(k'_i) = \sum_{z=0}^{t} a_z k'^{z}_i$



Considerations

$S = (s_1, \ldots, s_i, \ldots, s_n)$

$K = (k_1, \ldots, k_i, \ldots, k_n)$

$c = \text{store}_K(S)$

$K' = (H(k_1), \ldots, H(k_i), \ldots, H(k_n))$

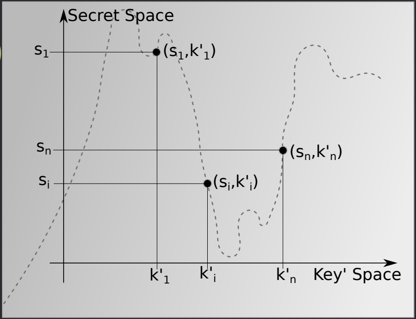
$c = \text{LagrangeInterpPol}_{K'}(S)$

$$c = \sum_{z=0}^{t} a_z x^z$$

$s_i = \text{retrieve}_{k_i}(c)$

$k'_i = H(k_i)$

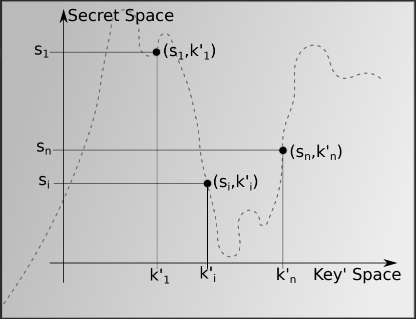
$$s_i = f(k'_i) = \sum_{z=0}^{t} a_z k'^z_i$$



Considerations

$$S = (s_1, \ldots, s_i, \ldots, s_n)$$
$$K = (k_1, \ldots, k_i, \ldots, k_n)$$
$$c = \text{store}_K(S)$$
$$K' = (H(k_1), \ldots, H(k_i), \ldots, H(k_n))$$
$$c = \text{LagrangeInterpPol}_{K'}(S)$$
$$c = \sum_{z=0}^{t} a_z x^z$$

$$s_i = \text{retrieve}_{k_i}(c)$$
$$k'_i = H(k_i)$$
$$s_i = f(k'_i) = \sum_{z=0}^{t} a_z {k'_i}^z$$



Considerations

Sometimes $\mathbb{F}_p$ is really small... Let's say $1000|K|$

Problem:

How can collisions of k' in small $\mathbb{F}_p$ be omitted

Solution:

Use a collision free mapping function per storage

$H_K : K \mapsto K' \in \mathbb{F}_p$ where $|K| = |K'|$

Considerations

Sometimes $\mathbb{F}_p$ is really small... Let's say $1000|K|$

Problem:
How can collisions of k' in small $\mathbb{F}_p$ be omitted

Solution:
Use a collision free mapping function per storage
$H_K : K \mapsto K' \in \mathbb{F}_p$ where $|K| = |K'|$

Considerations

Sometimes $\mathbb{F}_p$ is really small... Let's say $1000|K|$

Problem:
How can collisions of k' in small $\mathbb{F}_p$ be omitted

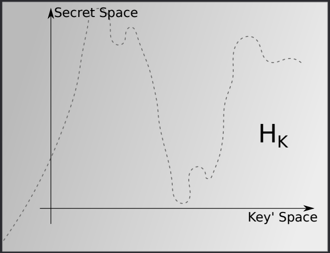
Solution:
Use a collision free mapping function per storage
$H_K : K \mapsto K' \in \mathbb{F}_p$ where $|K| = |K'|$

Considerations

$S = (s_1, \ldots, s_i, \ldots, s_n)$
$K = (k_1, \ldots, k_i, \ldots, k_n)$
$c = \text{store}_K(S)$
$H_K : K \mapsto K' \in \mathbb{F}_p || K| = |K'|$
$K' = (H_K(k_1), \ldots, H_K(k_i), \ldots, H_K(k_n))$
$c = \text{LagrangeInterpPol}_{K'}(S), H_K$
$c = \sum_{z=0}^{t} a_z x^z, \mathrm{H_K}$

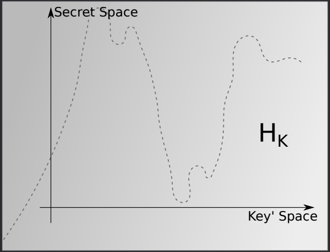
$s_i = \text{retrieve}_{k_i}(c)$
$k'_i = H_K(k_i)$

$s_i = f(k'_i) = \sum_{z=0}^{t} a_z k_i'^z$



Considerations

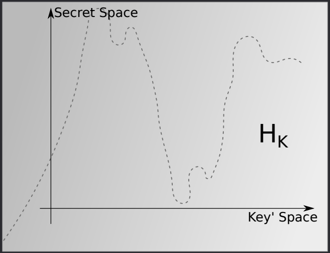
$$S = (s_1, \dots, s_i, \dots, s_n)$$
$$K = (k_1, \dots, k_i, \dots, k_n)$$
$$c = \text{store}_K(S)$$
$$H_K : K \mapsto K' \in \mathbb{F}_p \mid |K| = |K'|$$
$$K' = (H_K(k_1), \dots, H_K(k_i), \dots, H_K(k_n))$$
$$c = \text{LagrangeInterpPol}_{K'}(S), H_K$$
$$c = \sum_{z=0}^{t} a_z x^z, \text{H}_\text{K}$$
$$s_i = \text{retrieve}_{k_i}(c)$$
$$k'_i = H_K(k_i)$$
$$s_i = f(k'_i) = \sum_{z=0}^{t} a_z {k'_i}^z$$



Considerations

$$S = (s_1, \ldots, s_i, \ldots, s_n)$$
$$K = (k_1, \ldots, k_i, \ldots, k_n)$$
$$c = \text{store}_K(S)$$
$$H_K : K \mapsto K' \in \mathbb{F}_p \| K| = |K'|$$
$$K' = (H_K(k_1), \ldots, H_K(k_i), \ldots, H_K(k_n))$$
$$c = \text{LagrangeInterpPol}_{K'}(S), H_K$$
$$c = \sum_{z=0}^{t} a_z x^z, \mathrm{H_K}$$

$$s_i = \text{retrieve}_{k_i}(c)$$
$$k_i' = H_K(k_i)$$
$$s_i = f(k_i') = \sum_{z=0}^{t} a_z k_i'^z$$



Considerations

$S = (s_1, \ldots, s_i, \ldots, s_n)$

$K = (k_1, \ldots, k_i, \ldots, k_n)$

$c = \text{store}_K(S)$

$H_K : K \mapsto K' \in \mathbb{F}_p \| |K| = |K'|$

$K' = (H_K(k_1), \ldots, H_K(k_i), \ldots, H_K(k_n))$

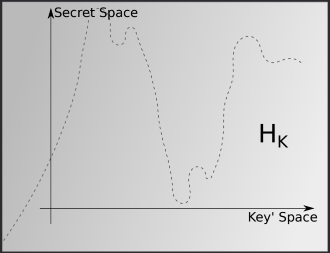
$c = \text{LagrangeInterpPol}_{K'}(S), H_K$

$$c = \sum_{z=0}^{t} a_z x^z, \mathrm{H_K}$$

$s_i = \text{retrieve}_{k_i}(c)$

$k'_i = H_K(k_i)$

$$s_i = f(k'_i) = \sum_{z=0}^{t} a_z {k'_i}^z$$



How to Use It For the 20-Credentials

A simple example (it works, but...)

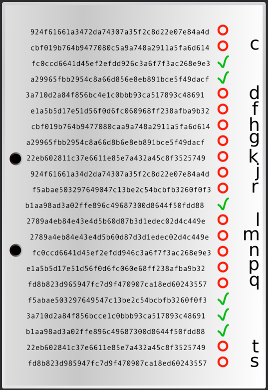
- ...use consonants as keys for the 'Drop' credentials
- ...use vowels as keys for 'Accept' credentials
- ...use some other keys for some credentials
- ...store them in the secret-storage system

924f61661a3472da74307a35f2c8d22e07e84a4d	○
cbf019b764b9477080c5a9a748a2911a5fa6d614	○
fc0ccd6641d45ef2efdd926c3a6f7f3ac268e9e3	✓
a29965fbb2954c8a66d856e8eb891bce5f49dacf	✓
3a710d2a84f856bc4e1c0bbb93ca517893c48691	○
e1a5b5d17e51d56f0d6fc060968ff238afba9b32	○
cbf019b764b9477080caa9a748a2911a5fa6d614	○
a29965fbb2954c8a66d8b6e8eb891bce5f49dacf	○
22eb602811c37e6611e85e7a432a45c8f3525749	○
924f61661a34d2da74307a35f2c8d22e07e84a4d	○
f5abae503297649047c13be2c54bcbfb3260f0f3	○
b1aa98ad3a02ffe896c49687300d8644f50fdd88	✓
2789a4eb84e43e4d5b60d87b3d1edec02d4c449e	○
2789a4eb84e43e4d5b60d87d3d1edec02d4c449e	○
fc0ccd6641d45ef2efdd946c3a6f7f3ac268e9e3	○
e1a5b5d17e51d56f0d6fc060e68ff238afba9b32	○
fd8b823d965947fc7d9f470907ca18ed60243557	○
f5abae503297649547c13be2c54bcbfb3260f0f3	✓
3a710d2a84f856bcce1c0bbb93ca517893c48691	✓
b1aa98ad3a02ffe896c49687300d8644f50fdd88	✓
22eb602841c37e6611e85e7a432a45c8f3525749	○
fd8b823d985947fc7d9f470907ca18ed60243557	○

How to Use It For the 20-Credentials

A simple example (it works, but...)

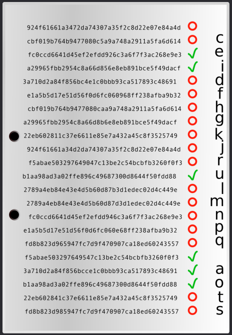
- ...use consonants as keys for the 'Drop' credentials
- ...use vowels as keys for 'Accept' credentials
- ...use some other keys for some credentials
- ...store them in the secret-storage system



How to Use It For the 20-Credentials

A simple example (it works, but...)

- ...use consonants as keys for the 'Drop' credentials
- ...use vowels as keys for 'Accept' credentials
- ...use some other keys for some credentials
- ...store them in the secret-storage system



How to Use It For the 20-Credentials

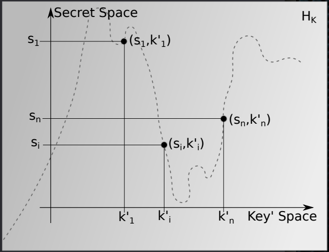
A simple example (it works, but...)

- ...use consonants as keys for the 'Drop' credentials
- ...use vowels as keys for 'Accept' credentials
- ...use some other keys for some credentials
- ...store them in the secret-storage system

How to Use It For the 20-Credentials

A simple example (it works, but...)

- ...use consonants as keys for the 'Drop' credentials
- ...use vowels as keys for 'Accept' credentials
- ...use some other keys for some credentials
- ...store them in the secret-storage system



The secret-storing system brings other very desirable features:

Typoo Resistance via Secret-Storing System | The OR-Function

Either key_i OR key_n unveils the single secret

$s_i = \text{retrieve}_{k_i}(c)$

$s_n = \text{retrieve}_{k_n}(c)$

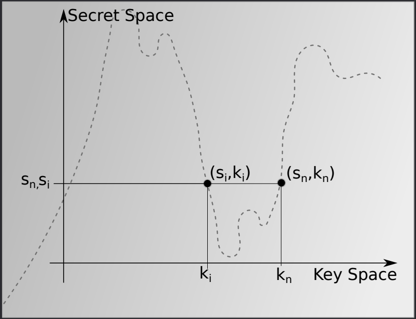
$verify(s_i = s_n)$

The secret-storing system brings other very desirable features:

Typoo Resistance via Secret-Storing System | The OR-Function

Either key_i OR key_n unveils the single secret

$$s_i = \text{retrieve}_{k_i}(c)$$
$$s_n = \text{retrieve}_{k_n}(c)$$
$$verify(s_i = s_n)$$



Key-Hinting via Secret-Storing System | The AND Function

Only key_i AND key_n unveil the single secret:

k_a	"HintMeWith...:"	s_a	"098z71adf4383498"
k_b	"GoodCredentials"	s_b	"aer9393fads932sv3"
k_c	"BadCredentials"	s_c	"598nnja2devm24v3a"
k_d	$s_a \oplus s_b$	s_d	"vowel"
k_e	$s_a \oplus s_c$	s_e	"consonant"

This still is secure as long as the keys carries some entropy!
It is definitely more secure than "What is your mother's maiden name" used nowadays!

And this one for free:

Secret-Sharing via Secret-Storing System | The THRESHOLD Function

Only n out of m keys unveil the single secret...

k_a	"Chief-1"	s_a	"098z71adf4383498"
k_b	"Chief-2"	s_b	"1gfnasdfrhhsadfn"
k_c	"Chief-3"	s_c	"fgjn439f3j22tj93"
k_x	$s_a \oplus s_b$	s_x	"We are bankrupt"
k_y	$s_a \oplus s_c$	s_y	"We are bankrupt"
k_z	$s_b \oplus s_c$	s_z	"We are bankrupt"

Randomized Secret-Storing System

If some data points R not representing not in (K, S) are chosen arbitrary, the system changes from a deterministic secret-storing system into a randomized secrets-storing System, where two stores containing the same (K, S) 'look' completely different.

Definition

A *randomized secret-storing system* of order n,

$$\Gamma = (\mathcal{S}, \mathcal{K}, \mathcal{R}, \mathcal{C}, \text{randomStore}, \text{retrieve}),$$

consists of a secret space $\mathcal{S}$, a key space $\mathcal{K}$, a randomization space $\mathcal{R}$, a storage space $\mathcal{C}$, and two functions randomStore and retrieve with properties as introduced above.

Randomized Secret-Storing System

If some data points R not representing not in (K, S) are chosen arbitrary, the system changes from a deterministic secret-storing system into a randomized secrets-storing System, where two stores containing the same (K, S) 'look' completely different.

Definition

A *randomized secret-storing system* of order n,

$$\Gamma = (\mathcal{S}, \mathcal{K}, \mathcal{R}, \mathcal{C}, \text{randomStore}, \text{retrieve}),$$

consists of a secret space $\mathcal{S}$, a key space $\mathcal{K}$, a randomization space $\mathcal{R}$, a storage space $\mathcal{C}$, and two functions randomStore and retrieve with properties as introduced above.

The Thing is a Beast

Field-homomorphism

Every operation (+,*) can be applied using two secret-stores as operands. The same operations will then be applied on all secrets of the same key.

This is exactly the most desired feature in order to work in a secure cloud system on the internet.

A CPU in the cloud can now do calculations using secret-stores, hence not knowing the operands nor the result.

Discussion

This is left as an exercise for the reader...

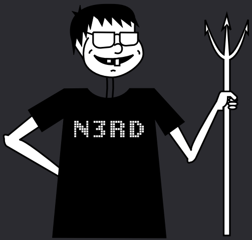
...One more Thing...

The Asymmetric variant of the secret-storing system with the same properties as shown above.

Please feel Free to Break the Secret-Storing System!

As it is a rather easy concept (even for a young math-student), it is easy to understand the system and thus to look for the show-stopper.... So, if you ever... Please let me know too!

Happy Hacking!



KryptonIT is Not Secure in Theory

The probability p that any $s_x \notin S$ matches to at least one $k_x \in K$ is equal to 0.
$p(s_x \notin S \text{ matches to at least one } k_x \notin K) = \frac{1}{|\mathcal{S}|}$.
$p(s_x \notin S \text{ matches to no } k_x \notin K) = 1 - \frac{1}{|\mathcal{S}|}$.
$p(|\mathcal{S}| s_x \notin S \text{ do not match any } k_x \notin K) = (1 - \frac{1}{|\mathcal{S}|})^{\mathcal{S}}$.
So: $\lim_{n\to\infty}(1 - \frac{1}{\mathcal{S}})^n = \frac{1}{e}$
This leads to the following entropy equation:
$H(s_x|c) = H(s_x) - \frac{1}{e}$

KryptonIT is Not Secure in Theory

The minuend of $H(s_x|c) = H(s_x) - \frac{1}{e}$ can be reduced if the mapping-space not not equal to the secret-space.
So K would not map to S directly but to M whereas M is a multiple ϕ of S.
A sub-space-mapping (modulo) then maps from M to S.
This leads to the following entropy equation:
$H(s_x|c) = H(s_x) - (\frac{1}{e})^{\frac{1}{\phi}}$
And...with $\lim_{\phi \to \infty}$
$H(s_x|c) = H(s_x)$

KryptonIT is Secure in Practical Usage

As long as $S \times \phi$ is chosen big enough ($>$200bit+(1bit per anno)) the system is considered to be secure in practice.
This is true as the analyses require to create a 'rainbow-table' for each c. This is considered infeasible for c spanning a big space.