Authenticating compromisable storage systems

Jiangshan Yu
University of Luxembourg, Luxembourg
University of Birmingham, UK

Joint work with Mark Ryan and Liqun Chen

Aim:
Backup sensitive data in the cloud.

Aim:
Backup sensitive data in the cloud.

Problem:
How to secure the data?

Aim:
Backup sensitive data in the cloud.

Problem:
How to secure the data?

Solution:
Encrypt data before uploading it.

Aim:
Backup sensitive data in the cloud.

Problem:
How to secure the data?

Solution:
Encrypt data before uploading it.

Problem Cont.:
How to secure the key?

Aim:
Backup sensitive data in the cloud.

Problem:
How to secure the data?

Solution:
Encrypt data before uploading it.

Problem Cont.:
How to secure the key?

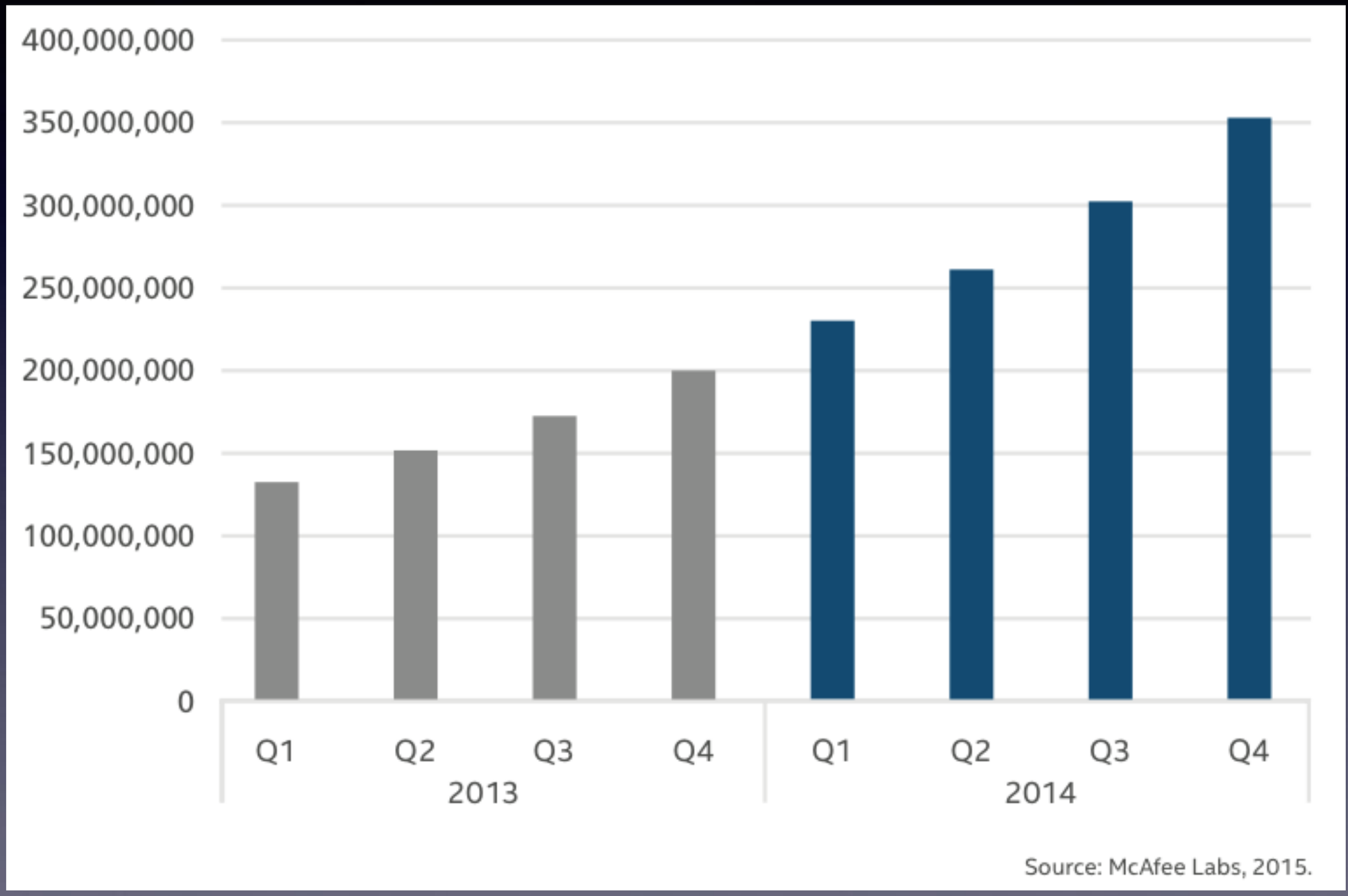
Solution Cont.:
Store it locally!

Operating systems with most security vulnerabilities

Operating system	# of vulnerabilities	# of HIGH vulnerabilities	# of MEDIUM vulnerabilities	# of LOW vulnerabilities
Apple Mac OS X	147	64	67	16
Apple iOS	127	32	72	23
Linux Kernel	119	24	74	21
Microsoft Windows Server 2008	38	26	12	0
Microsoft Windows 7	36	25	11	0
Microsoft Windows Server 2012	38	24	14	0
Microsoft Windows 8	36	24	12	0
Microsoft Windows 8.1	36	24	12	0
Microsoft Windows Vista	34	23	11	0
Microsoft Windows RT	30	22	8	0

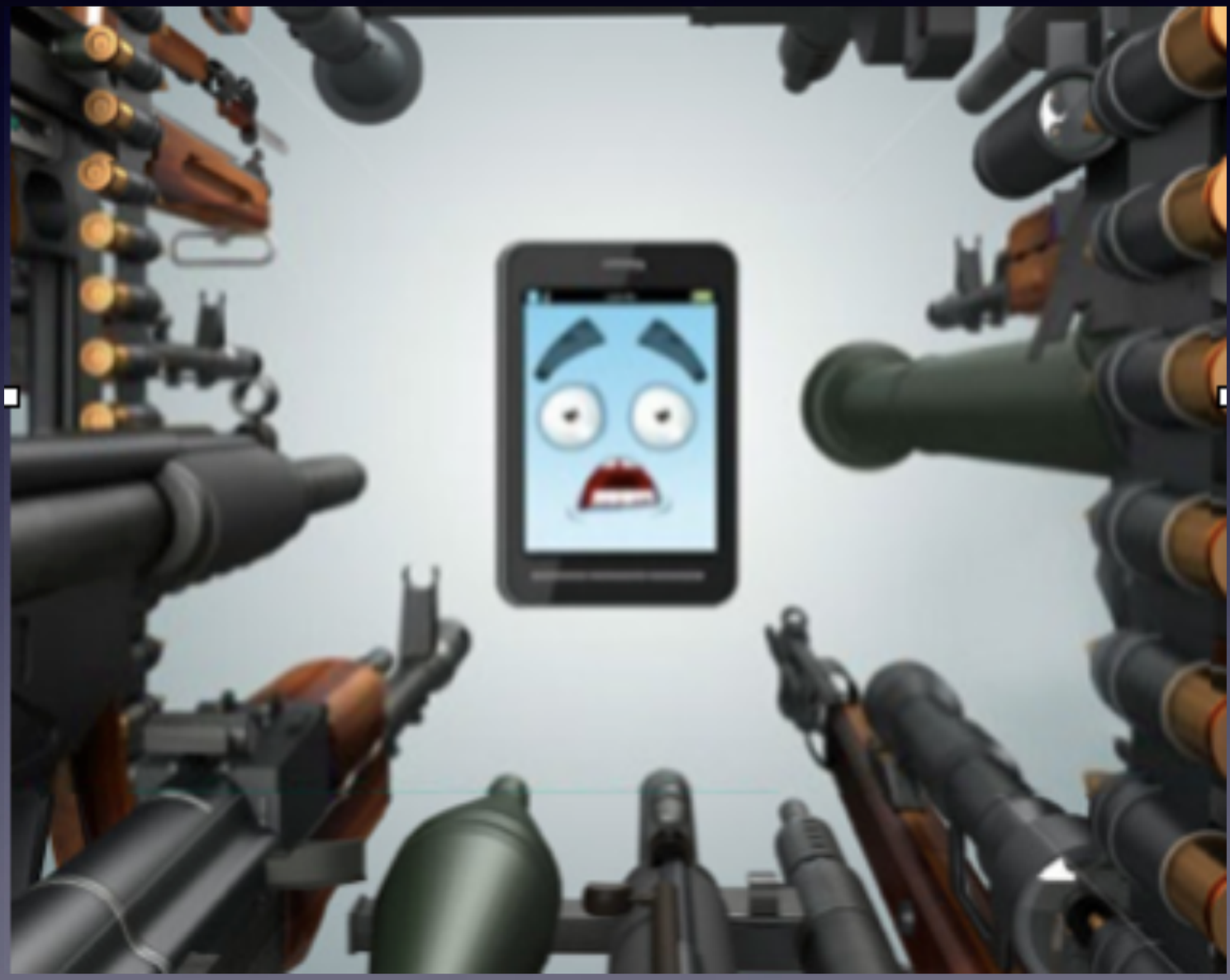
http://www.gfi.com/blog/most-vulnerable-operating-systems-and-applications-in-2014/

Total number of malware instances collected by McAfee Labs across PCs and Mobile devices



Source: McAfee Labs, 2015.

Your device is not secure!



Aim:
Backup sensitive data in the cloud.

Problem:
How to secure the data?

Solution:
Encrypt data before upload

Problem Cont.:
How to secure the key?

~~Solution Cont.:~~
~~Store it locally!~~

Aim:
Backup sensitive data in the cloud.

Problem:
How to secure the data?

Solution:
Encrypt data before upload

Problem Cont.:
How to secure the key?

~~Solution Cont.:~~
~~Store it locally!~~

Solution Cont.:
(k,n)-secret sharing in different platforms!

(k,n)-secret sharing cryptosystem:

- Distribute a secret amongst n platforms;
- Each platform obtains a share of the secret;
- A secret can only be reconstructed if at least k shares are collected.

We hope that in the **life time** of the system, the attacker cannot compromise k servers.

Murphy’s law:
Anything that can go wrong will go wrong.

What could we do?

Our Goal: make an attacker's job more difficult.

Attacker’s work

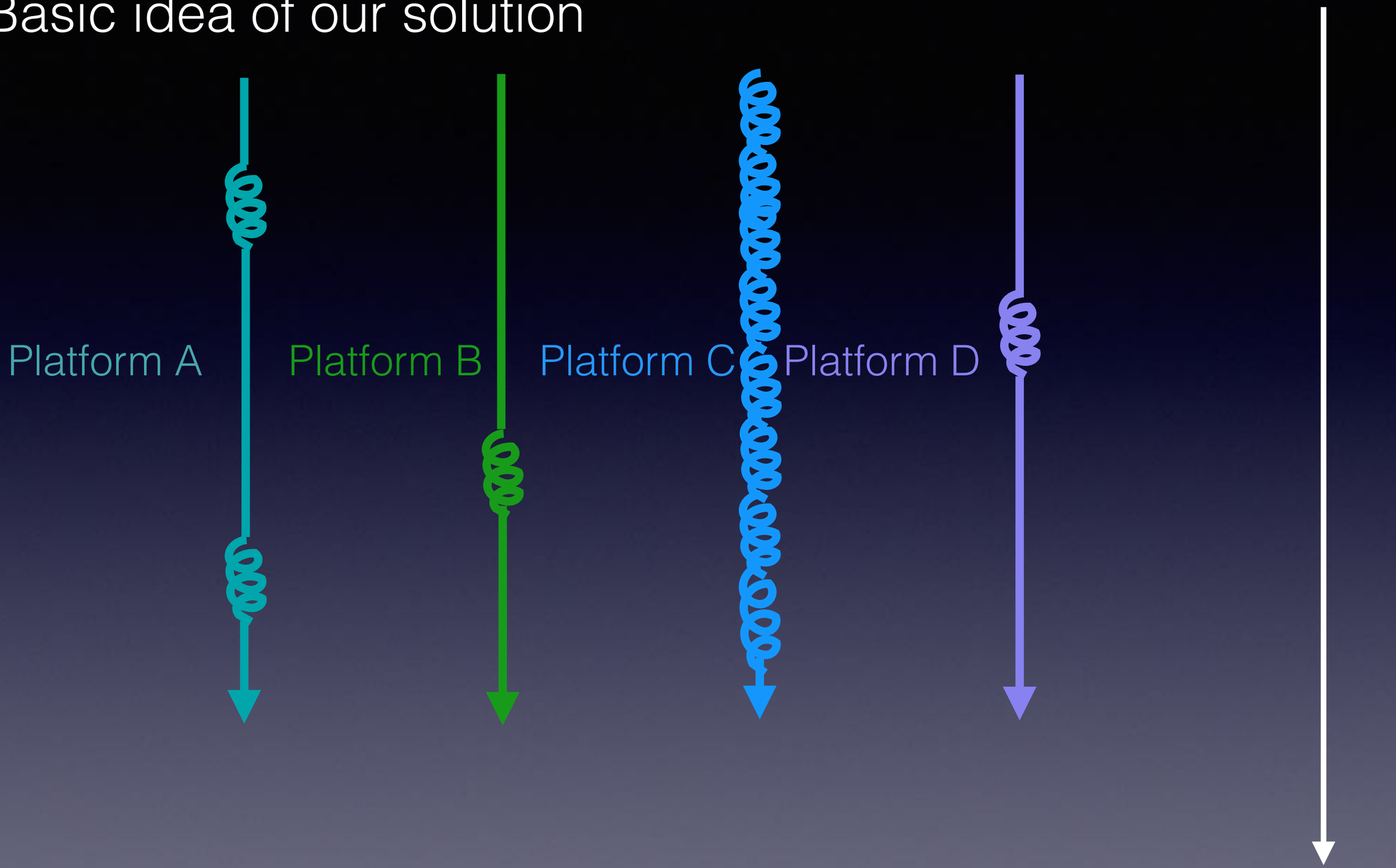
Compromise **k** platforms in the **life time** of the system

Attacker's work

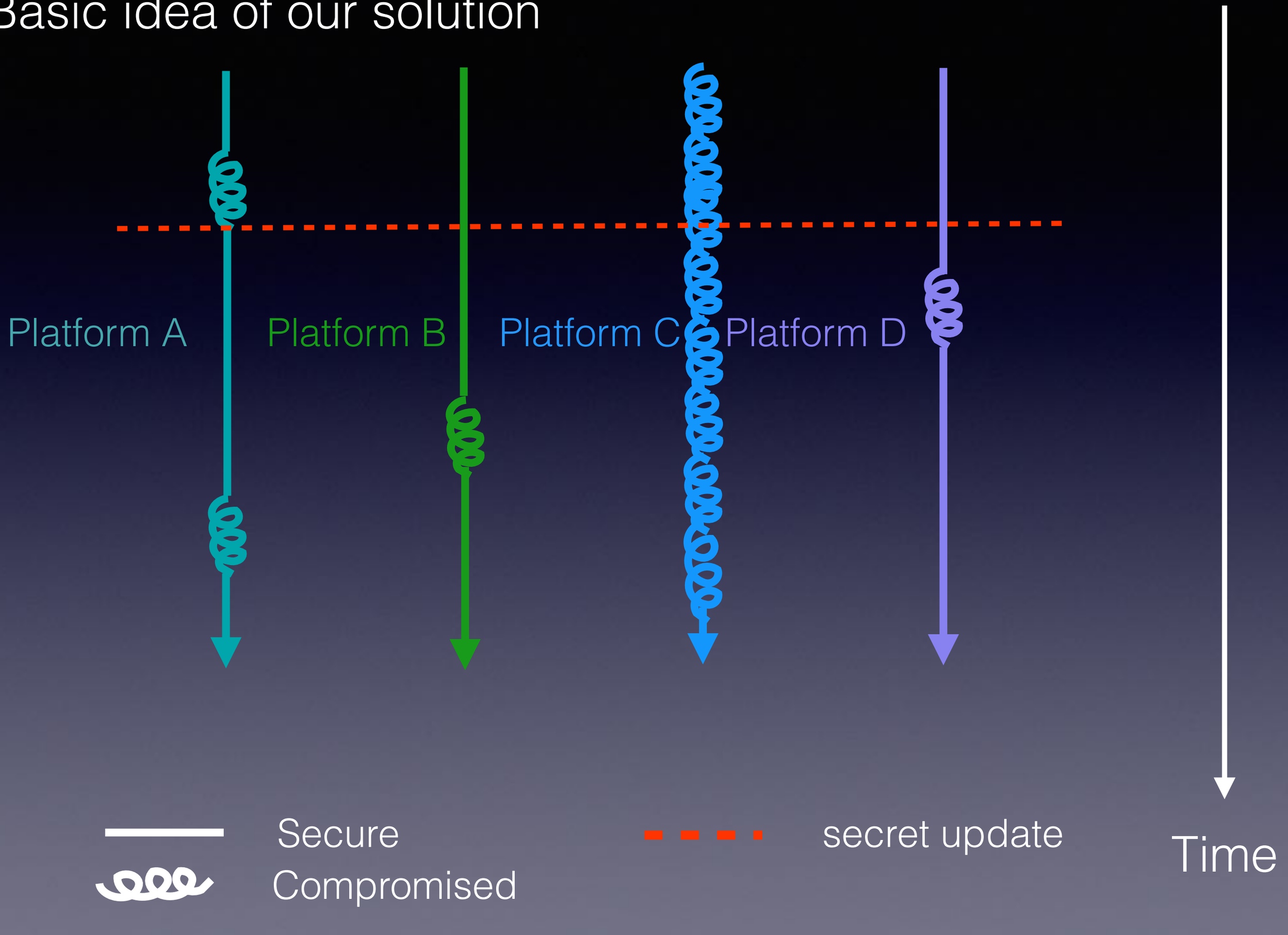
Compromise **k** platforms in the ~~**life time**~~ **time** of the system

a **short period**

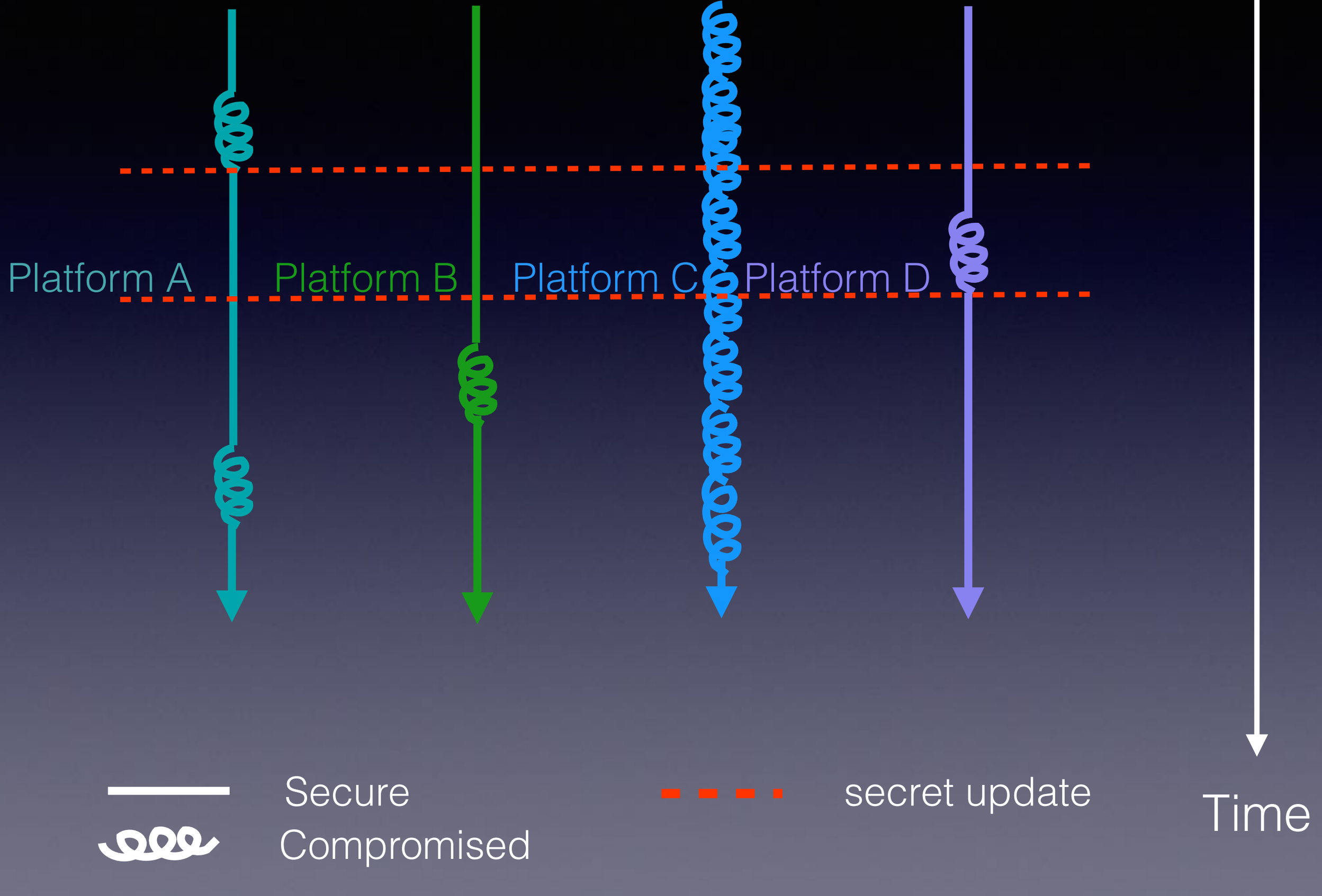
Basic idea of our solution



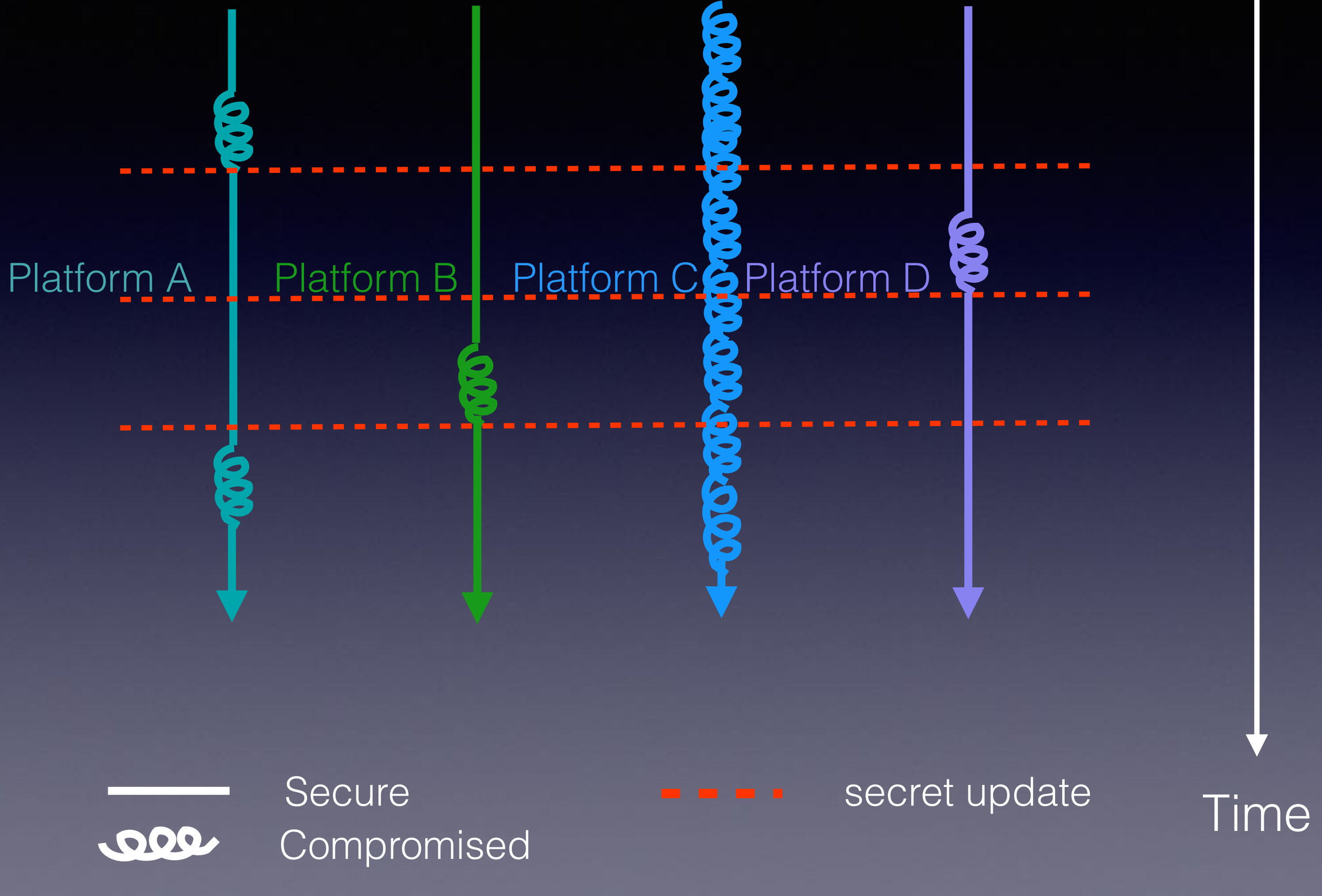
Basic idea of our solution



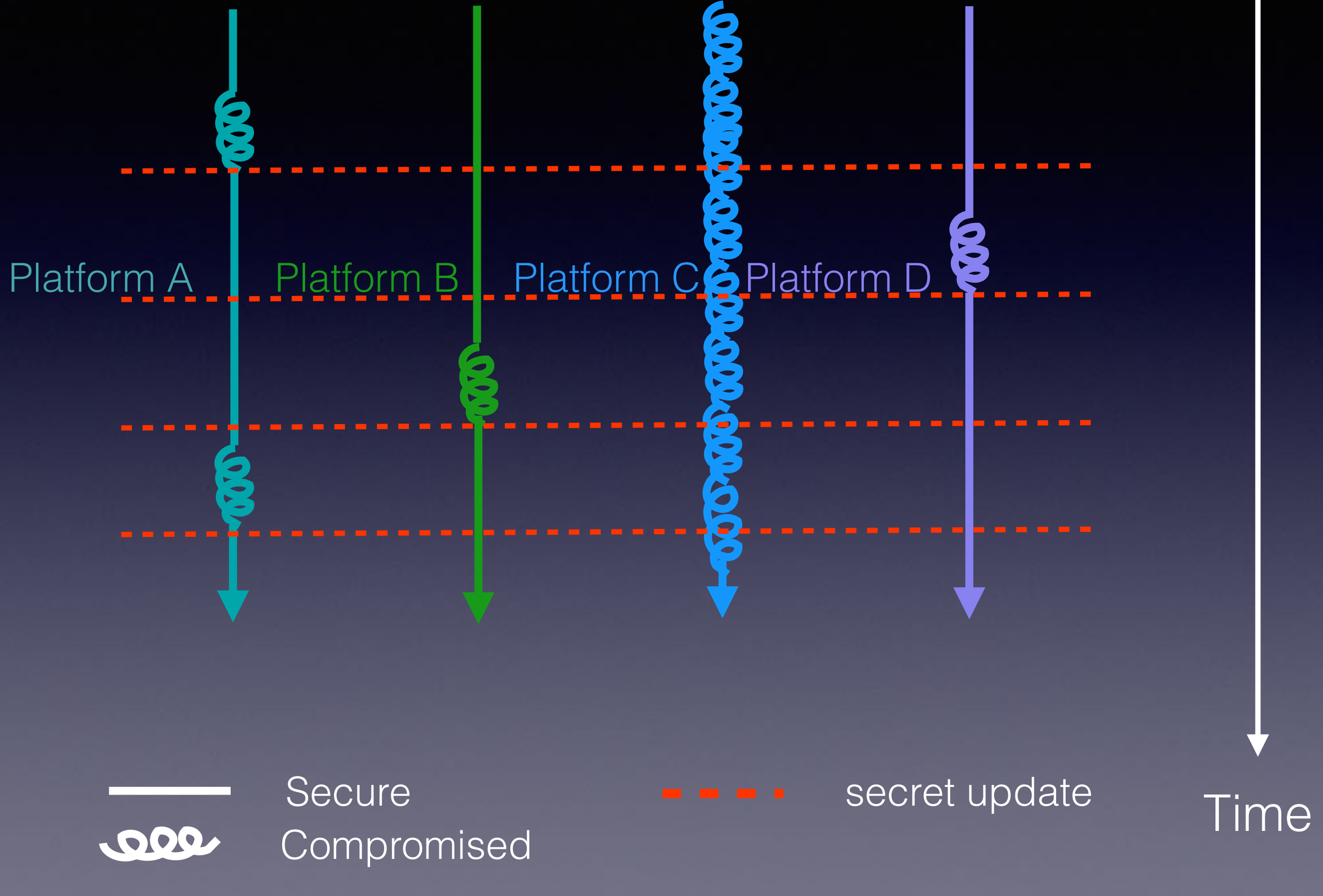
Basic idea of our solution



Basic idea of our solution



Basic idea of our solution



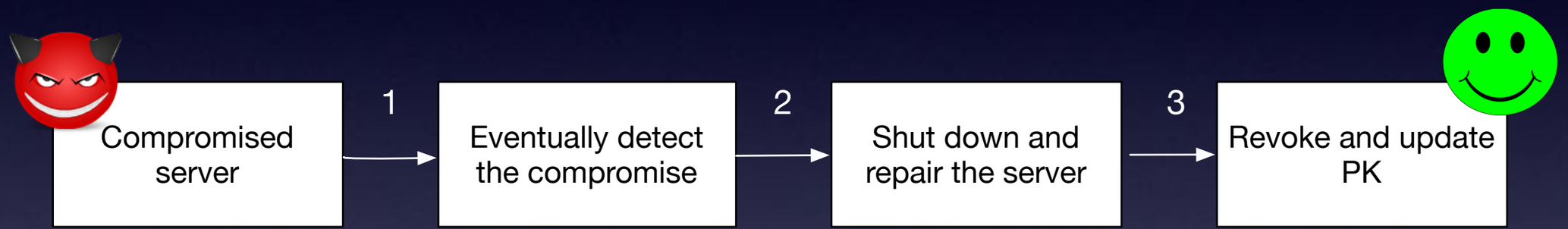
Challenges

1. How to authenticate a server if it has already been compromised?

Challenges

1. How to authenticate a server if it has already been compromised?

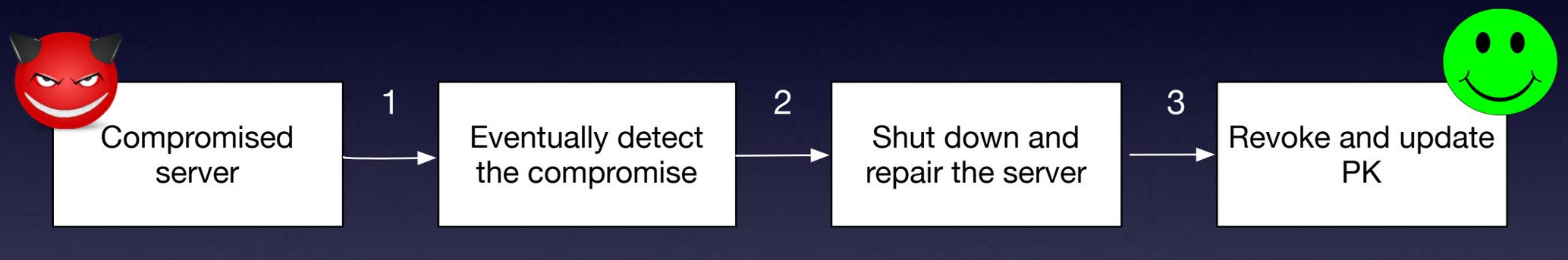
   Currently:



Challenges

1. How to authenticate a server if it has already been compromised?

   Currently:

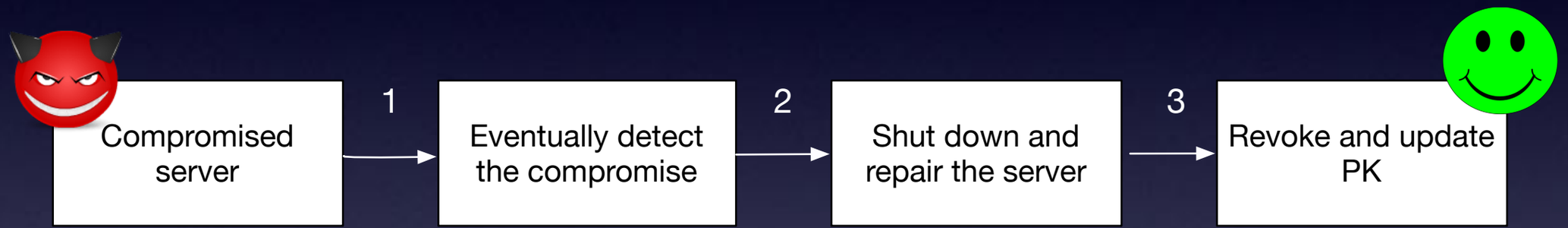


   However, PKI has its own security problem and several compromised CAs are discovered in recent years.
   [Yu et. al, Computer Journal, 2016; Yu and Ryan, SABDC, chapter 7, 2017]

Challenges

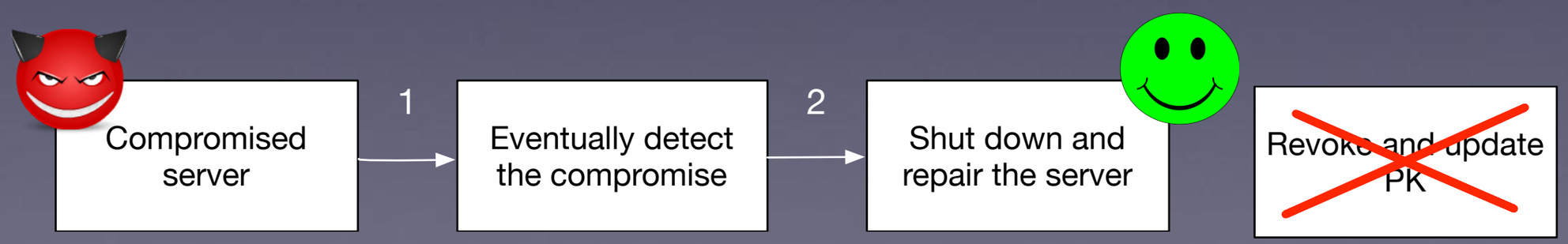
1. How to authenticate a server if it has already been compromised?

   Currently:



   However, PKI has its own security problem and several compromised CAs are discovered in recent years.
   [Yu et. al, Computer Journal, 2016; Yu and Ryan, SABDC, chapter 7, 2017]

   Goal: A fixed group PK to avoid revoking and re-distributing PK



Challenges

1. How to authenticate a server if the server has already been compromised?

Our solution

- A constant group public key that will not change

Challenges

1. How to authenticate a server if the server has already been compromised?
2. How to update secret keys without changing the corresponding group PK?

Our solution

- A constant group public key that will not change

Challenges

1. How to authenticate a server if the server has already been compromised?
2. How to update an SK without changing the corresponding PK?
3. How to avoid having users' involvement in the secret update?

Our solution

- A constant group public key that will not change

Challenges

1. How to authenticate a server if the server has already been compromised?
2. How to update an SK without changing the corresponding PK?
3. How to avoid having users' involvement in the secret update?

Our solution

- A constant group public key that will not change
- We designed a new cryptographic system based on Bilinear paring to provide these features.

Challenges

1. How to authenticate a server if the server has already been compromised?
2. How to update an SK without changing the corresponding PK?
3. How to avoid having users' involvement in every secret update?
4. How do you know if the new system is secure?

Our solution

- A constant group public key that will not change
- We designed a new cryptographic system based on Bilinear paring to provide these features.

Challenges

1. How to authenticate a server if the server has already been compromised?
2. How to update an SK without changing the corresponding PK?
3. How to avoid having users' involvement in every secret update?
4. How do you know if the new system is secure?

Our solution

- A constant group public key that will not change
- We designed a new cryptographic system based on Bilinear paring to provide these features.
- We define a formal model of the system, and provide formal security proofs.

Protocol (simplified)

Initialisation

G_1, G_2: groups of prime order q
$e : G_1^2 \to G_2$
$g \in G_1$ and $Z = e(g, g) \in G_2$ are random generators.

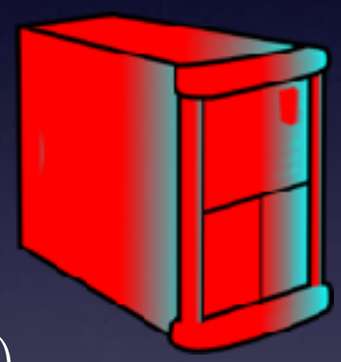
Protocol (simplified)

Initialisation

G_1, G_2: groups of prime order q
$e : G_1^2 \to G_2$
$g \in G_1$ and $Z = e(g, g) \in G_2$ are random generators.

(a, g^a)

(a_1, g^{a_1})



(b, g^b)

(b_1, g^{b_1})

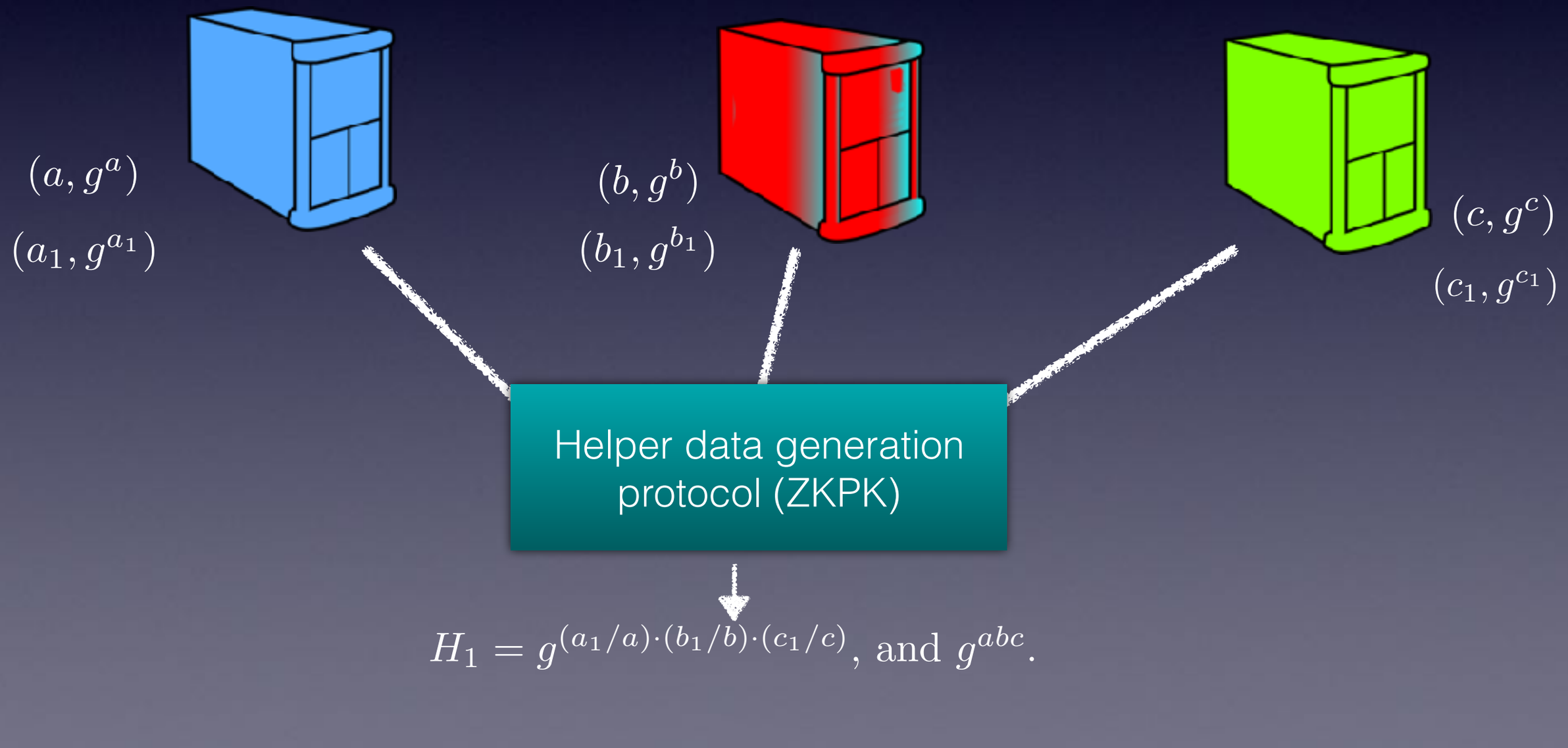
(c, g^c)

(c_1, g^{c_1})

Protocol (simplified)

Initialisation

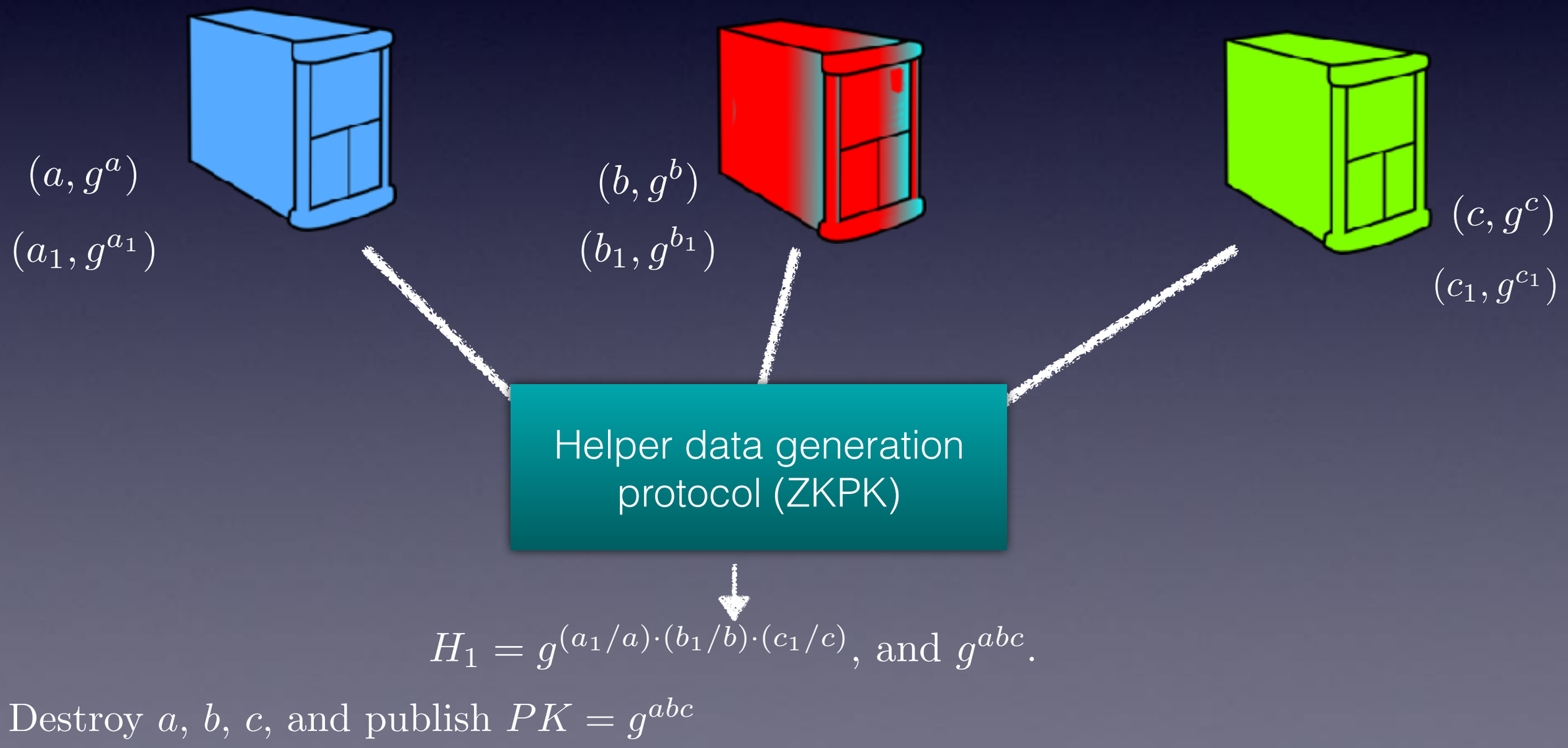
G_1, G_2: groups of prime order q
$e : G_1^2 \to G_2$
$g \in G_1$ and $Z = e(g, g) \in G_2$ are random generators.



Protocol (simplified)

Initialisation

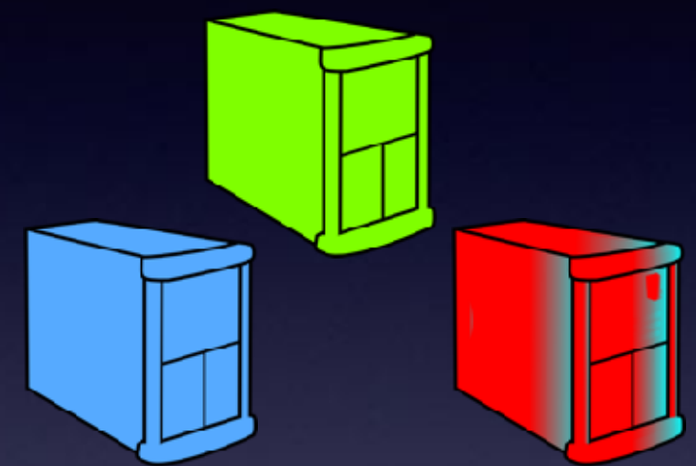
G_1, G_2: groups of prime order q
$e : G_1^2 \to G_2$
$g \in G_1$ and $Z = e(g, g) \in G_2$ are random generators.



$H_1 = g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)}$, and g^{abc}.

Destroy a, b, c, and publish $PK = g^{abc}$

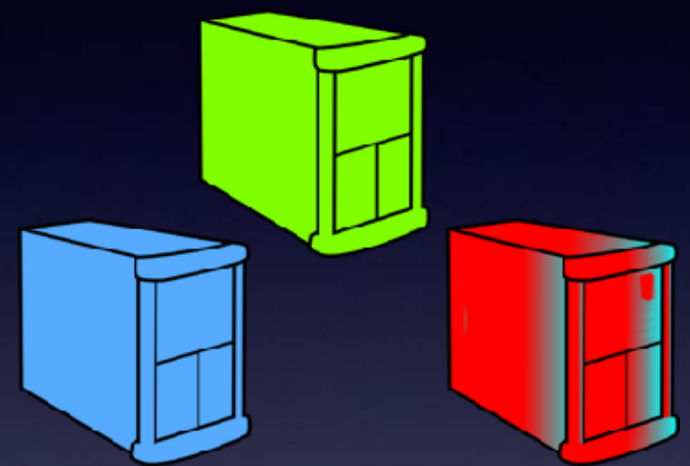
Encryption



- generate $k \in \mathbb{Z}_q$
- compute PK^k and sZ^k
- compute proof P of knowledge about k

Encryption

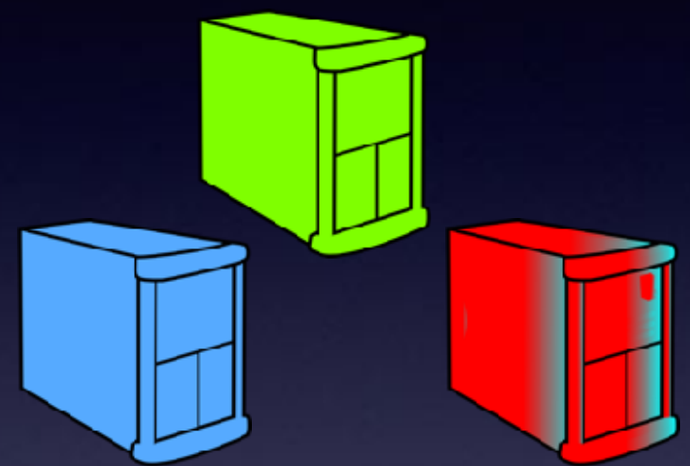
$(\alpha = PK^k, \beta = sZ^k, P)$



- generate $k \in \mathbb{Z}_q$
- compute PK^k and sZ^k
- compute proof P of knowledge about k

Encryption

$(\alpha = PK^k, \beta = sZ^k, P)$

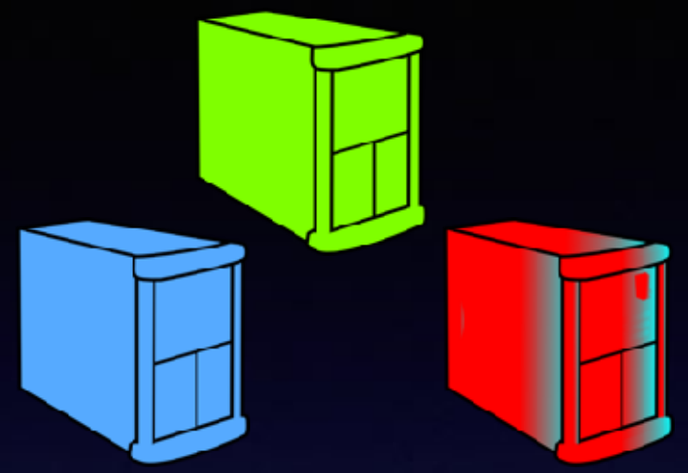


- generate $k \in \mathbb{Z}_q$
- compute PK^k and sZ^k
- compute proof P of knowledge about k

- verify P
- store (α, β)

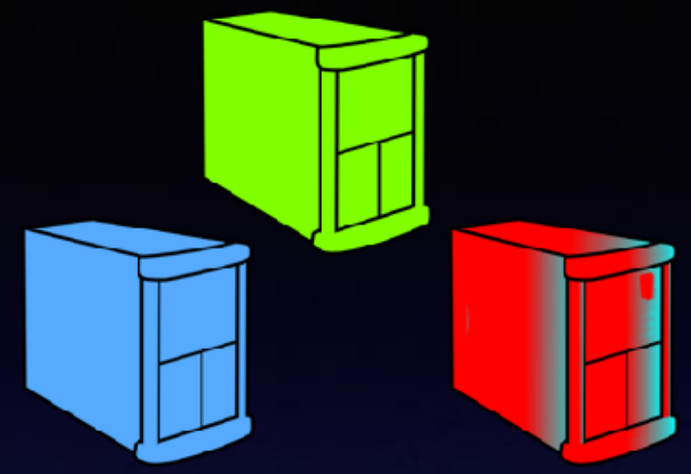
Decryption

User authentication



Decryption

User authentication

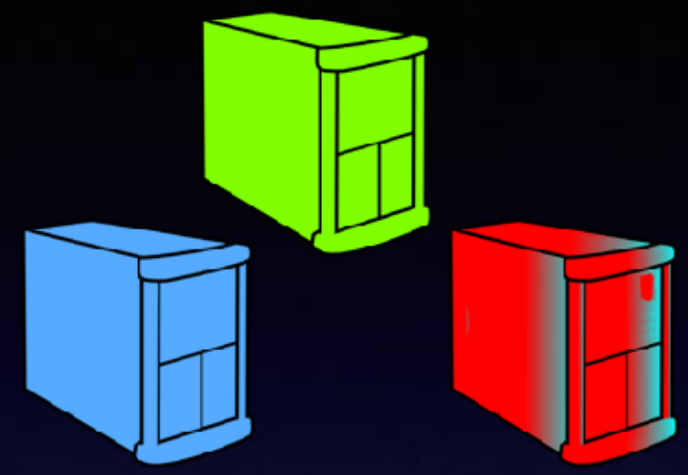


- calculate γ, s.t.

$$\gamma = e(\alpha, H_1)$$
$$= e(g^{abck}, g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)})$$
$$= Z^{a_1 b_1 c_1 k}$$

Decryption

User authentication



$(\beta, \gamma) = (sZ^k, Z^{a_1 b_1 c_1 k})$

- calculate γ, s.t.

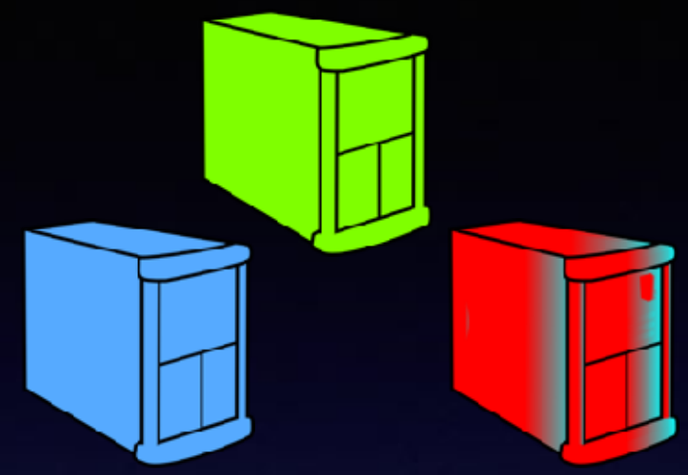
$\gamma = e(\alpha, H_1)$

$= e(g^{abck}, g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)})$

$= Z^{a_1 b_1 c_1 k}$

Decryption

User authentication



$(\beta, \gamma) = (sZ^k, Z^{a_1 b_1 c_1 k})$

- calculate γ, s.t.

$\gamma = e(\alpha, H_1)$

$= e(g^{abck}, g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)})$

$= Z^{a_1 b_1 c_1 k}$

- generate $k' \in \mathbb{Z}_q$
- computes $\gamma^{k'} = Z^{a_1 b_1 c_1 k k'}$

$\gamma^{k'}$

Decryption

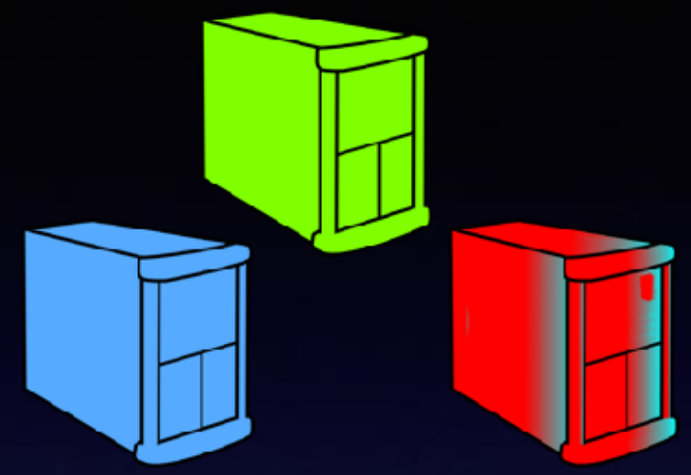
User authentication

$(\beta, \gamma) = (sZ^k, Z^{a_1 b_1 c_1 k})$

- generate $k' \in \mathbb{Z}_q$
- computes $\gamma^{k'} = Z^{a_1 b_1 c_1 k k'}$

$\gamma^{k'}$

$Z^{kk'}$



- calculate γ, s.t.

$\gamma = e(\alpha, H_1)$

$= e(g^{abck}, g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)})$

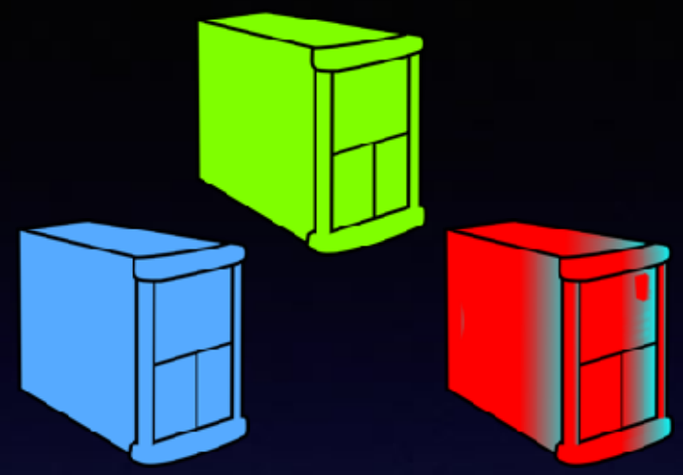
$= Z^{a_1 b_1 c_1 k}$

Remove $a_1 b_1 c_1$ from the exponent

Decryption

User authentication

$(\beta, \gamma) = (sZ^k, Z^{a_1 b_1 c_1 k})$



- calculate γ, s.t.

$\gamma = e(\alpha, H_1)$

$= e(g^{abck}, g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)})$

$= Z^{a_1 b_1 c_1 k}$

- generate $k' \in \mathbb{Z}_q$
- computes $\gamma^{k'} = Z^{a_1 b_1 c_1 k k'}$

$\gamma^{k'}$

Remove $a_1 b_1 c_1$ from the exponent

$Z^{kk'}$

- recover Z^k
- decrypt $\beta = sZ^k$

Updating keys and ciphertext

$sk_a = a_1$, $PK = g^{abc}$
$H_1 = g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)}$ and
a set of (α, β)

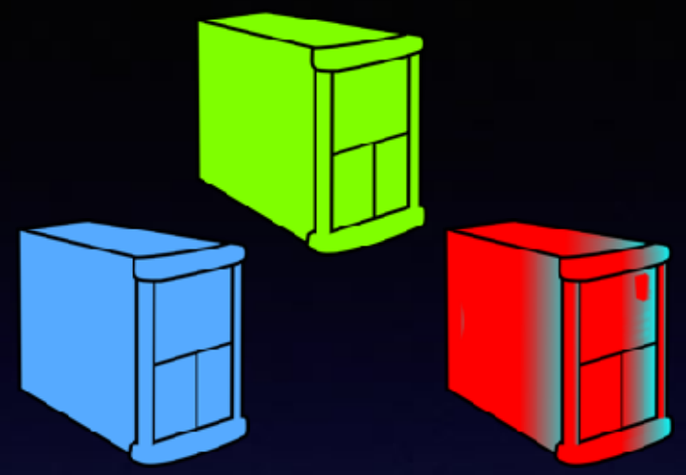
- generate a_2
- compute $H_2 = H_1^{(a_2/a_1)\cdot(b_2/b_1)\cdot(c_2/c_1)} = g^{(a_2/a)\cdot(b_2/b)\cdot(c_2/c)}$ with other servers
- destroy old a_1
- replace H_1 by using H_2

*No need to update any ciphertext.

Decryption

User authentication

$(\beta, \gamma) = (sZ^k, Z^{a_1 b_1 c_1 k})$



- calculate γ, s.t.

$\gamma = e(\alpha, H_1)$

$= e(g^{abck}, g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)})$

$= Z^{a_1 b_1 c_1 k}$

- generate $k' \in \mathbb{Z}_q$
- computes $\gamma^{k'} = Z^{a_1 b_1 c_1 k k'}$

$\gamma^{k'}$

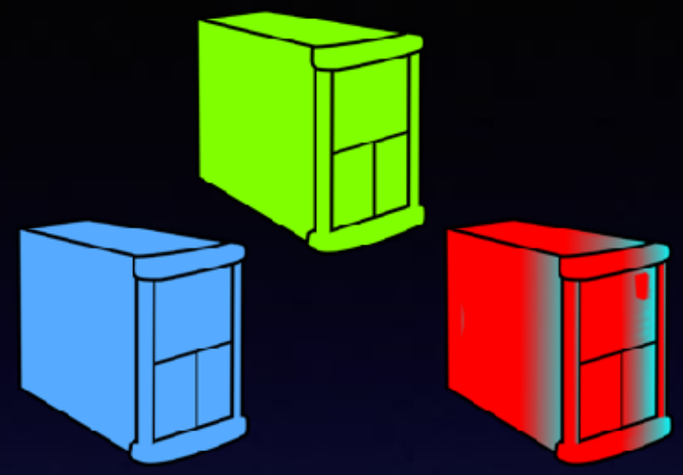
$Z^{kk'}$

Remove $a_1 b_1 c_1$ from the exponent

- recover Z^k
- decrypt $\beta = sZ^k$

Decryption

User authentication



$(\beta, \gamma) = (sZ^k, Z^{a_1 b_1 c_1 k})$

- calculate γ, s.t.

$$\gamma = e(\alpha, H_1)$$
$$= e(g^{abck}, g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)})$$
$$= Z^{a_1 b_1 c_1 k}$$

- generate $k' \in \mathbb{Z}_q$
- computes $\gamma^{k'} = Z^{a_1 b_1 c_1 k k'}$

$\gamma^{k'}$

$Z^{kk'}$

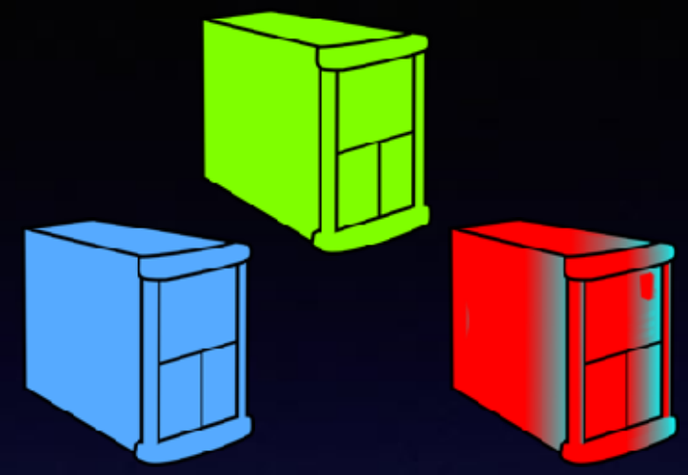
$\gamma = e(\alpha, H_2)$
$= e(g^{abck}, g^{(a_2/a)(b_2/b)(c_2/c)})$
$= Z^{a_2 b_2 c_2 k}$

Remove $a_1 b_1 c_1$ from the exponent

- recover Z^k
- decrypt $\beta = sZ^k$

Decryption

User authentication



$(\beta, \gamma) = (sZ^k, Z^{a_1 b_1 c_1 k})$

- calculate γ, s.t.

$$\gamma = e(\alpha, H_1)$$
$$= e(g^{abck}, g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)})$$
$$= Z^{a_1 b_1 c_1 k}$$

- generate $k' \in \mathbb{Z}_q$
- computes $\gamma^{k'} = Z^{a_1 b_1 c_1 k k'}$

$\gamma^{k'} = Z^{a_2 b_2 c_2 k k'}$

$\gamma = e(\alpha, H_2)$
$= e(g^{abck}, g^{(a_2/a)(b_2/b)(c_2/c)})$
$= Z^{a_2 b_2 c_2 k}$

$\gamma^{k'}$

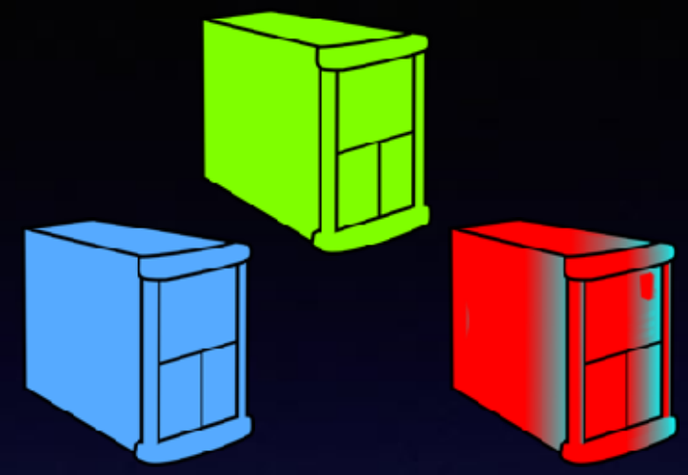
$Z^{kk'}$

Remove $a_1 b_1 c_1$ from the exponent

- recover Z^k
- decrypt $\beta = sZ^k$

Decryption

User authentication



$(\beta, \gamma) = (sZ^k, Z^{a_1 b_1 c_1 k})$

- calculate γ, s.t.

$$\begin{aligned}\gamma &= e(\alpha, H_1) \\ &= e(g^{abck}, g^{(a_1/a)\cdot(b_1/b)\cdot(c_1/c)}) \\ &= Z^{a_1 b_1 c_1 k}\end{aligned}$$

- generate $k' \in \mathbb{Z}_q$
- computes $\gamma^{k'} = Z^{a_1 b_1 c_1 k k'}$

$\gamma^{k'} = Z^{a_2 b_2 c_2 k k'}$

$\gamma = e(\alpha, H_2)$

$= e(g^{abck}, g^{(a_2/a)(b_2/b)(c_2/c)})$

$= Z^{a_2 b_2 c_2 k}$

$\gamma^{k'}$

$Z^{kk'}$

Remove $a_1 b_1 c_1$ from the exponent

$a_2 b_2 c_2$

- recover Z^k
- decrypt $\beta = sZ^k$

Recap:

- A scheme to update secret keys without changing the group PK
- For each update, the ciphertext remains the same
- No-user involvement in the key update process

Thanks!