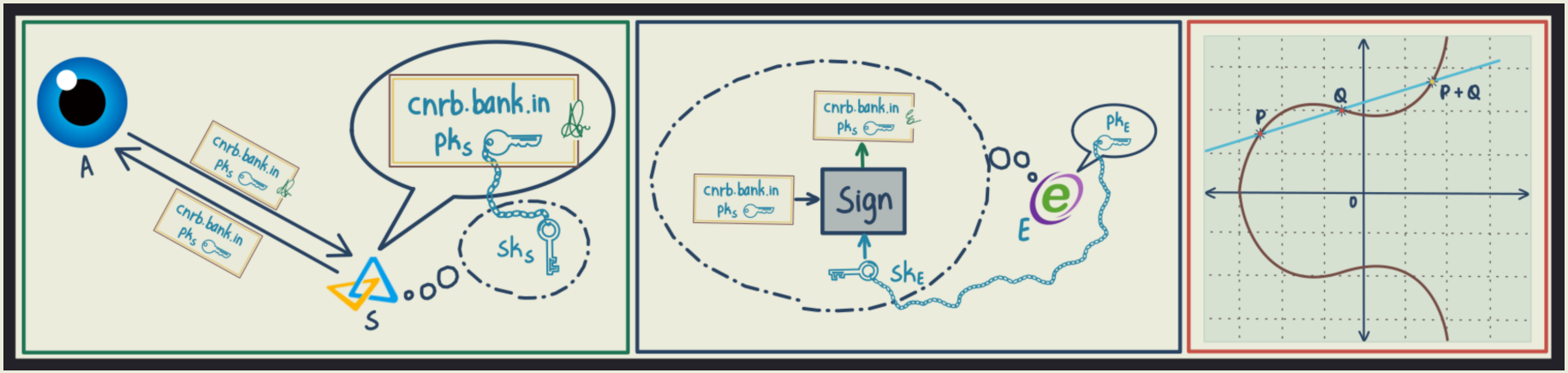


What Really... Happened at eMudhra?

Chethan Kamath

IIT Bombay

TL

informal

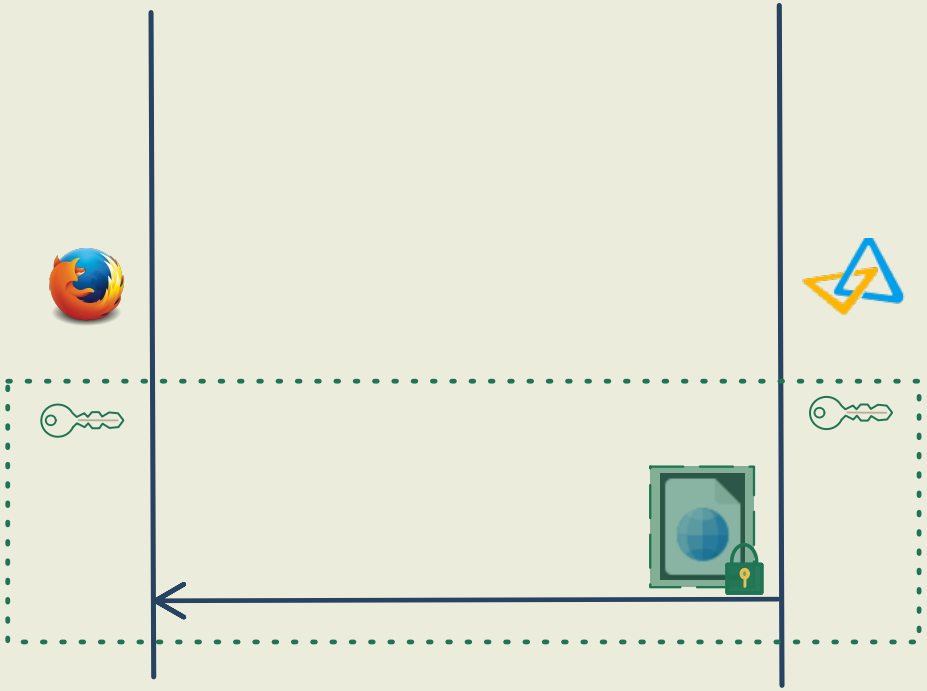
Our Plan

1. Part I: How Identity and Trust Works Over the Internet
2. Part II: What Went Wrong at eMudhra
3. Part III: The Aftermath

Our Plan

1. Part I: How Identity and Trust Works Over the Internet
2. Part II: What Went Wrong at eMudhra
3. Part III: The Aftermath

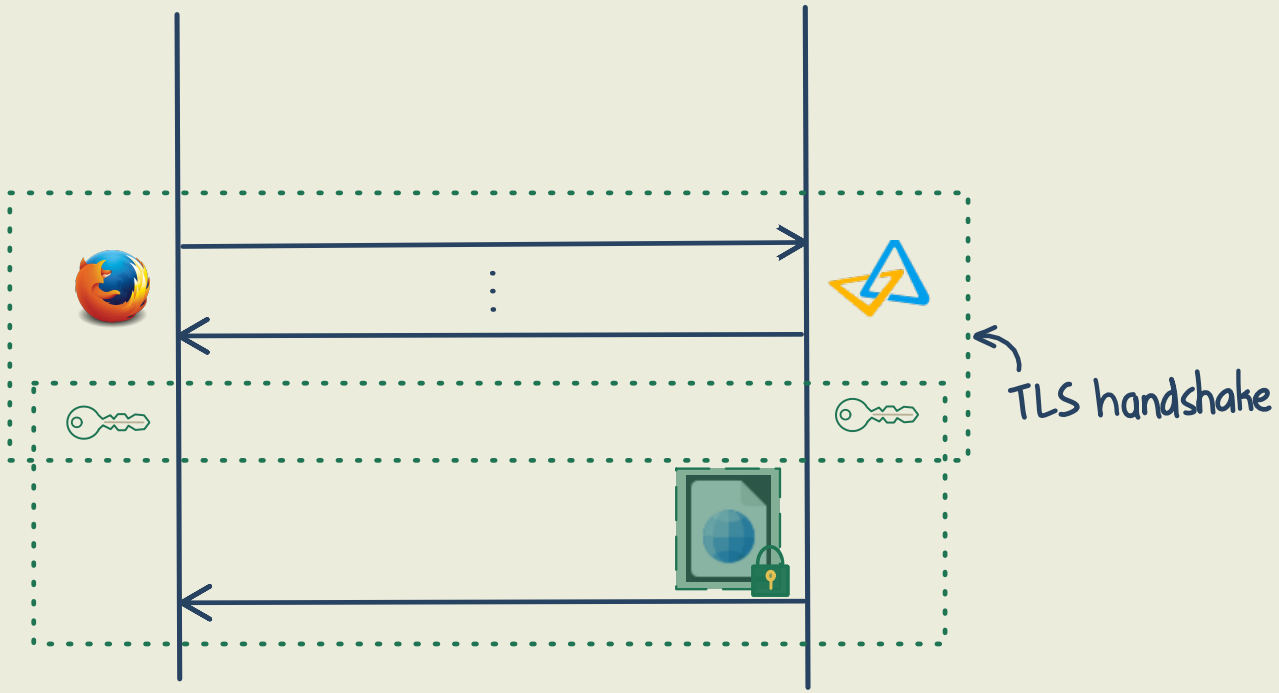
What Happens When You Access Netbanking?



1.  and  need a (symmetric) key to communicate securely

All logos were obtained from Wikipedia.org

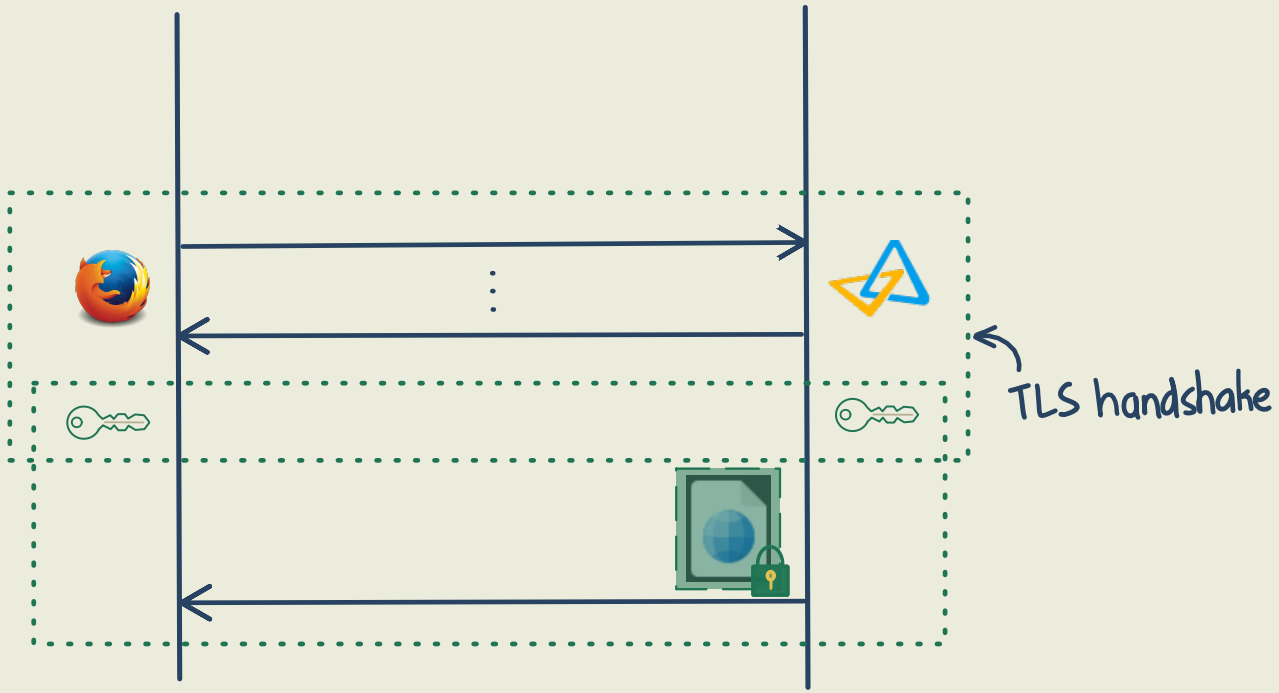
What Happens When You Access Netbanking?



1. and need a (symmetric) key to communicate securely
2. and must first establish this key *over the internet*

All logos were obtained from Wikipedia.org

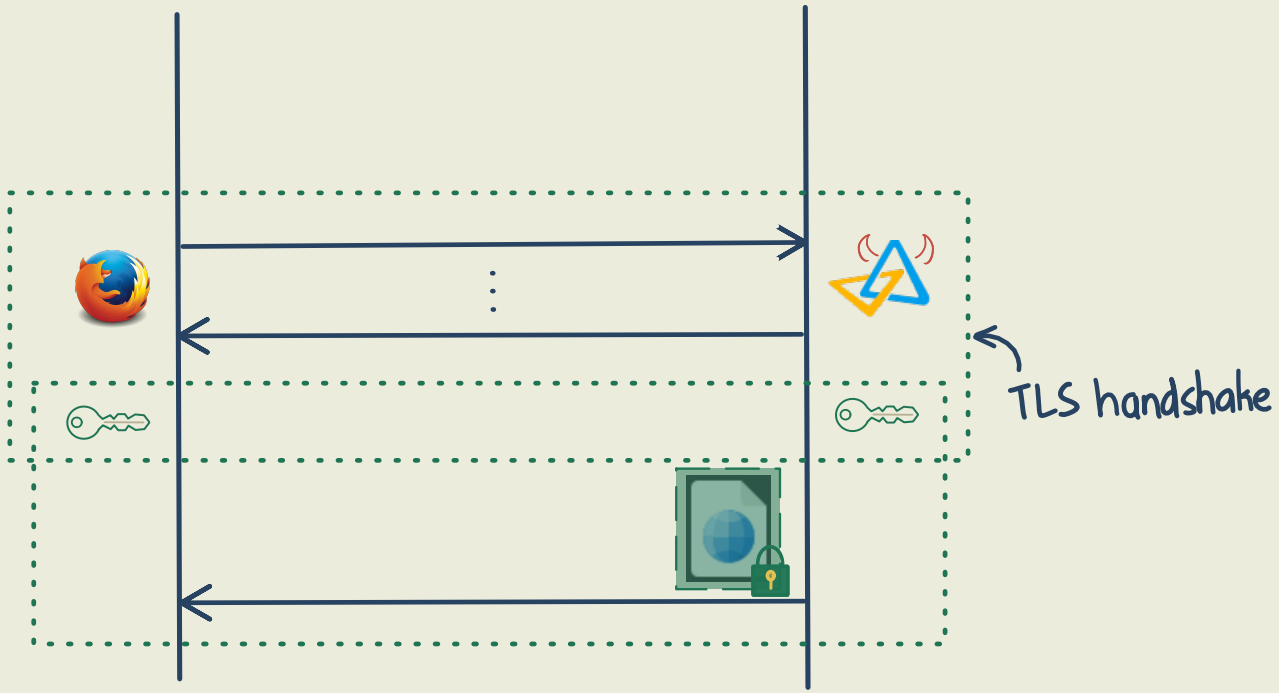
What Happens When You Access Netbanking?



1. Firefox and Cloudflare need a (symmetric) key to communicate securely
2. Firefox and Cloudflare must first establish this key *over the internet*
3. Firefox must ensure it is indeed talking to Cloudflare

All logos were obtained from Wikipedia.org

What Happens When You Access Netbanking?



1. Firefox and Cloudflare need a (symmetric) key to communicate securely
2. Firefox and Cloudflare must first establish this key *over the internet*
3. Firefox must ensure it is indeed talking to Cloudflare and not a phishing website trying to steal your password!

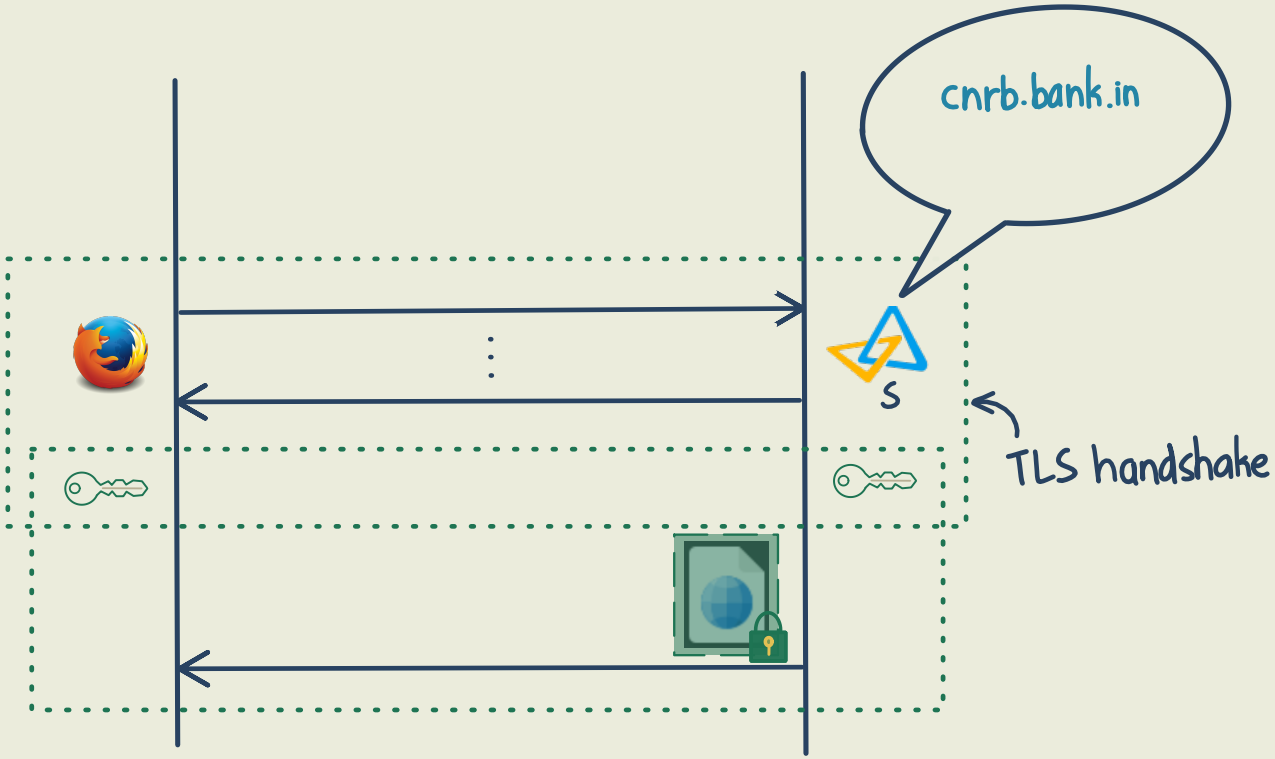
All logos were obtained from Wikipedia.org

What Constitutes Identity Over the Internet?



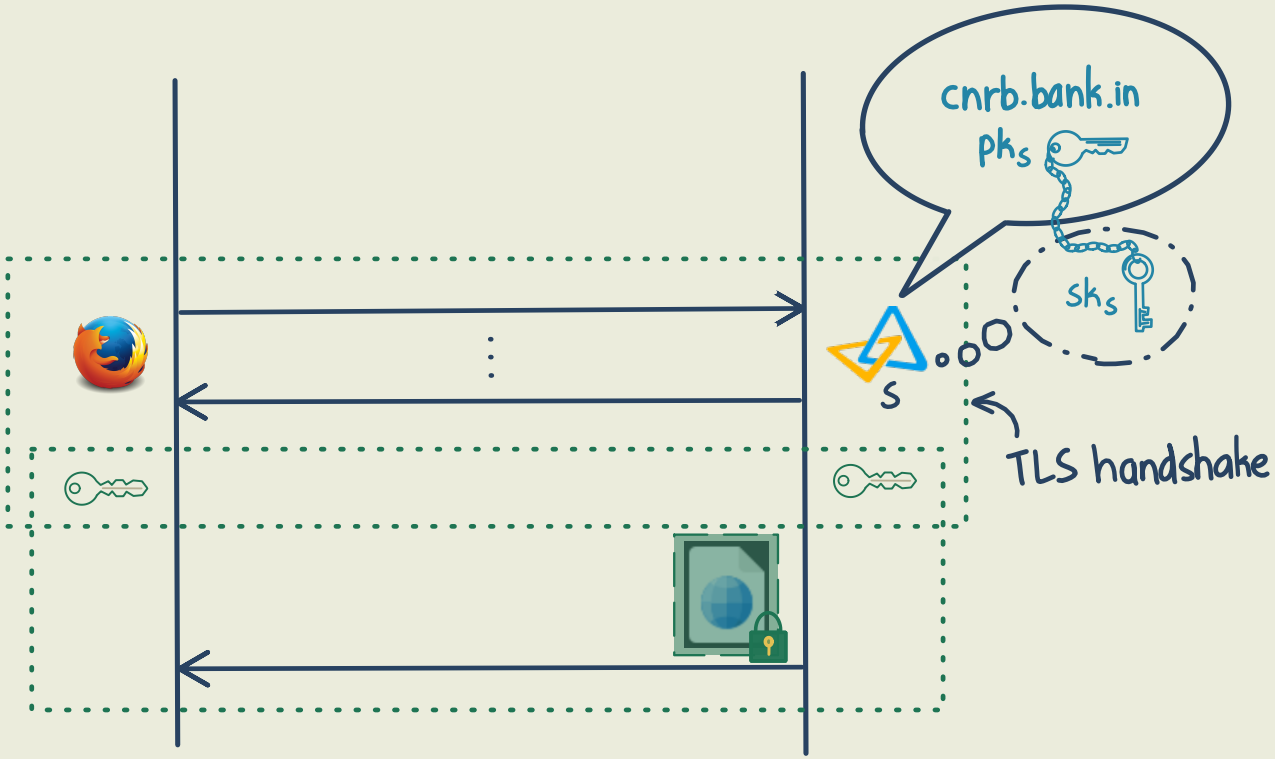
©New Yorker

What Constitutes Identity Over the Internet?



For communicating in the clear: S's domain name $\texttt{dn}_S$ (say)

What Constitutes Identity Over the Internet?

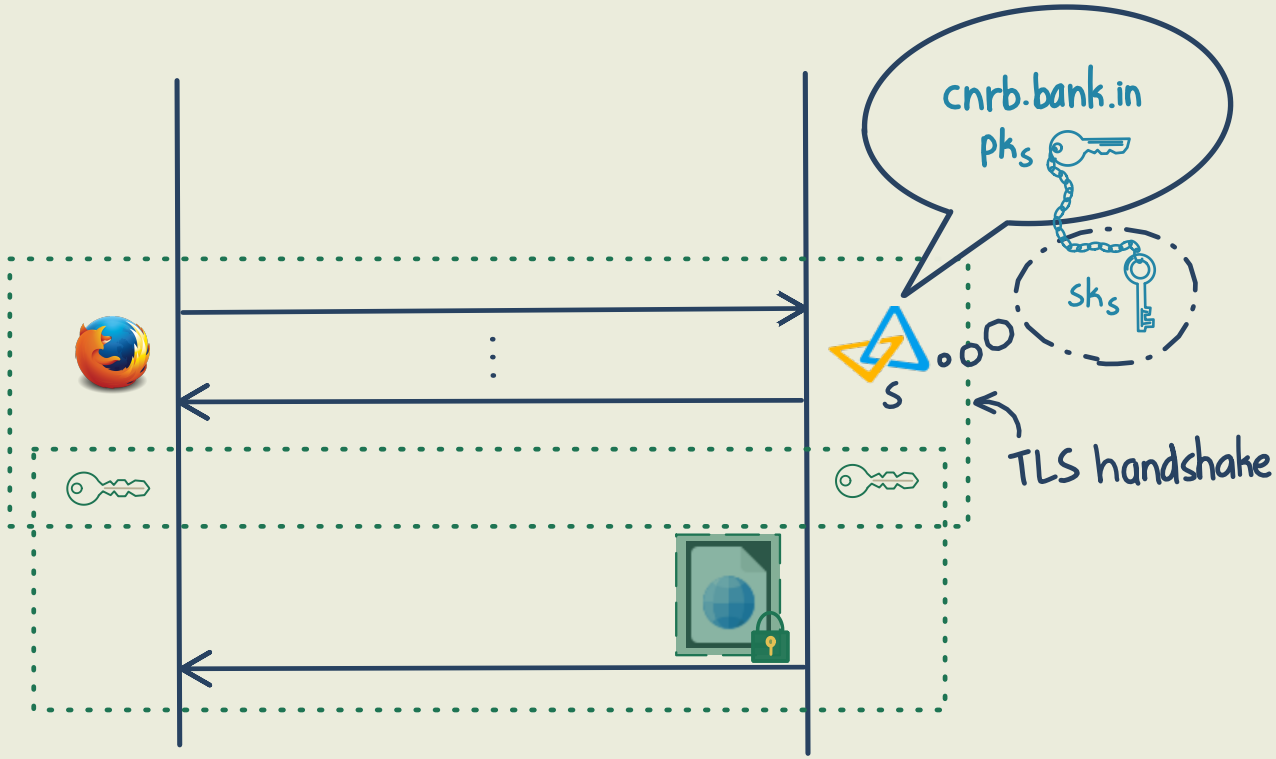


For communicating in the clear: S's domain name $\mathtt{dn}_S$ (say)

For communicating *securely*: *public key* (a cryptographic identity)

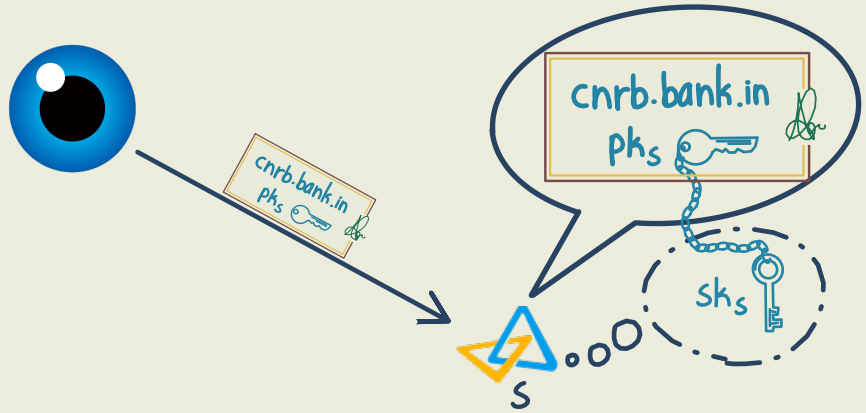
- S generates $\mathtt{pk}_S$ along with a *secret key* $\mathtt{sk}_S$
- S must keep $\mathtt{sk}_S$ hidden for $\mathtt{pk}_S$ to have any credence

What Constitutes Identity Over the Internet?



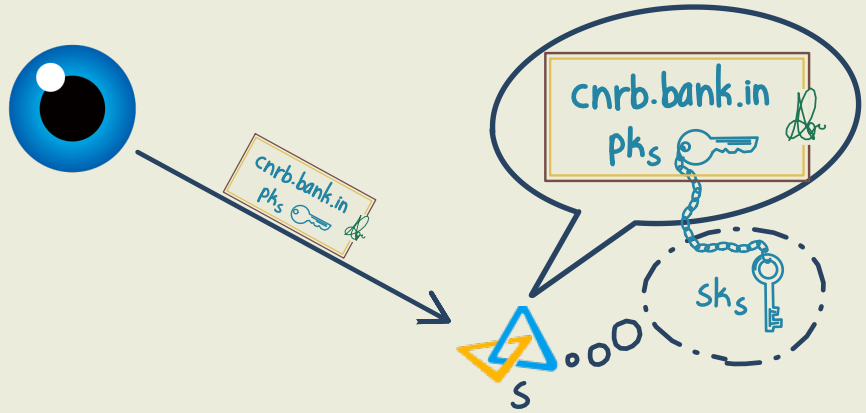
- For communicating in the clear: S's domain name $\mathtt{dn}_S$ (say)
- For communicating *securely*: *public key* (a cryptographic identity)
  - S generates $\mathtt{pk}_S$ along with a *secret key* $\mathtt{sk}_S$
  - S must keep $\mathtt{sk}_S$ hidden for $\mathtt{pk}_S$ to have any credence
- How to "bind" a domain name $\mathtt{dn}_S$ to a public key $\mathtt{pk}_S$?

Enter Certificates and Certifying Authorities (CAs)



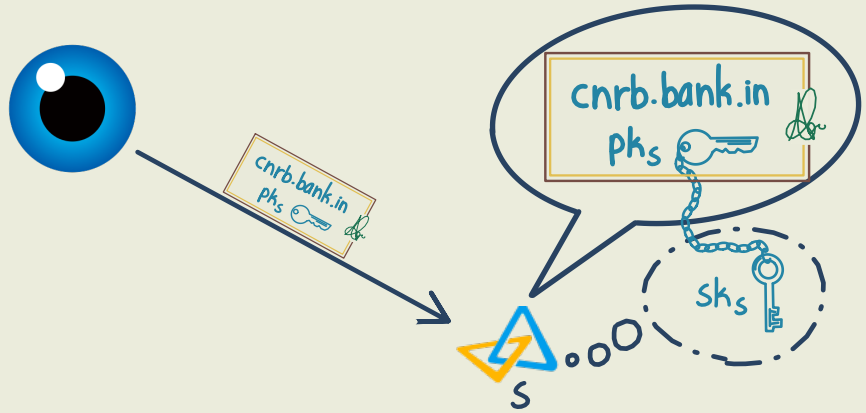
- Certificate: document binding domain name $\mathtt{dn}_S$ to public key $\mathtt{pk}_S$
- CA: *trusted third party* who "issues" certificates

Enter Certificates and Certifying Authorities (CAs)



- Certificate: document binding domain name $\texttt{dn}_S$ to public key $\texttt{pk}_S$
- CA: *trusted third party* who "issues" certificates
  - E.g.: eMudhra, Let' Encrypt, GlobalSign...

Enter Certificates and Certifying Authorities (CAs)

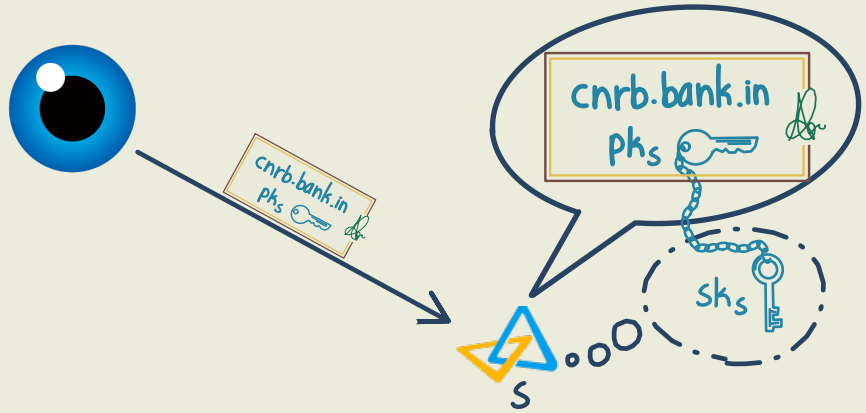


- Certificate: document binding domain name $\mathtt{dn}_S$ to public key $\mathtt{pk}_S$
- CA: *trusted third party* who "issues" certificates
  - E.g.: eMudhra, Let' Encrypt, GlobalSign...

What are the requirements?

1. CAs must *vet the subject* of the certificate (i.e., S) before issuing

Enter Certificates and Certifying Authorities (CAs)

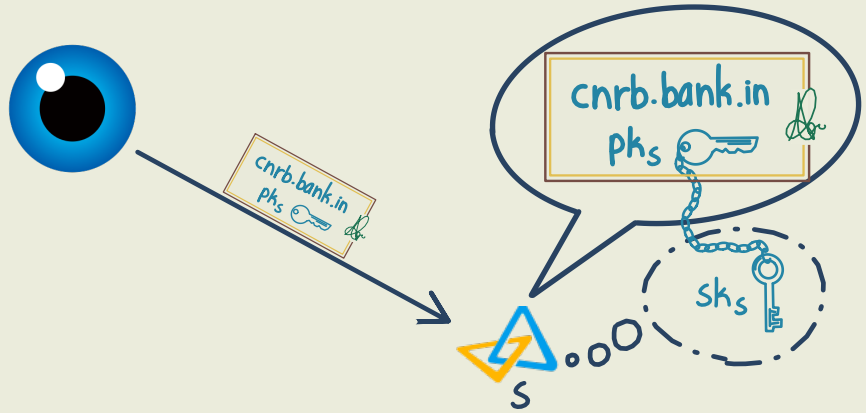


- Certificate: document binding domain name $\mathtt{dn}_S$ to public key $\mathtt{pk}_S$
- CA: *trusted third party* who "issues" certificates
  - E.g.: eMudhra, Let' Encrypt, GlobalSign...

What are the requirements?

1. CAs must *vet the subject* of the certificate (i.e., S) before issuing
2. Certificate must be *publicly verifiable*

Enter Certificates and Certifying Authorities (CAs)



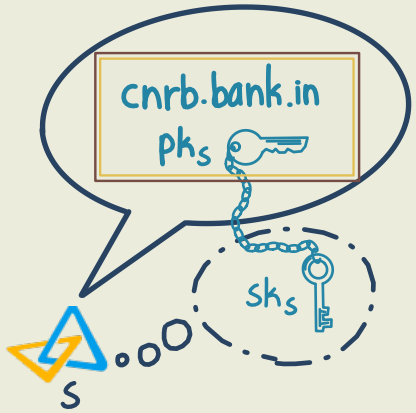
- Certificate: document binding domain name $\mathtt{dn}_S$ to public key $\mathtt{pk}_S$
- CA: *trusted third party* who "issues" certificates
  - E.g.: eMudhra, Let' Encrypt, GlobalSign...

What are the requirements?

1. CAs must *vet the subject* of the certificate (i.e., S) before issuing
2. Certificate must be *publicly verifiable*
3. Must not be possible to issue certificates *pretending to be* a CA

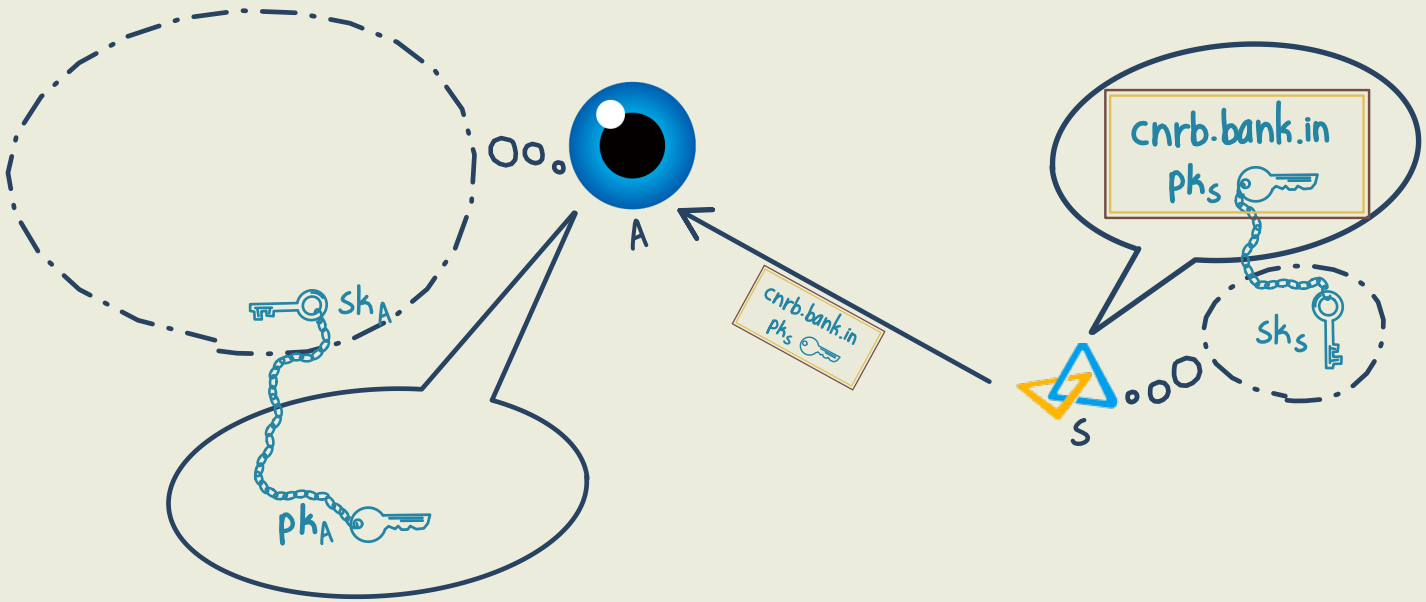
How Exactly does a CA *A* Issue a Certificate?





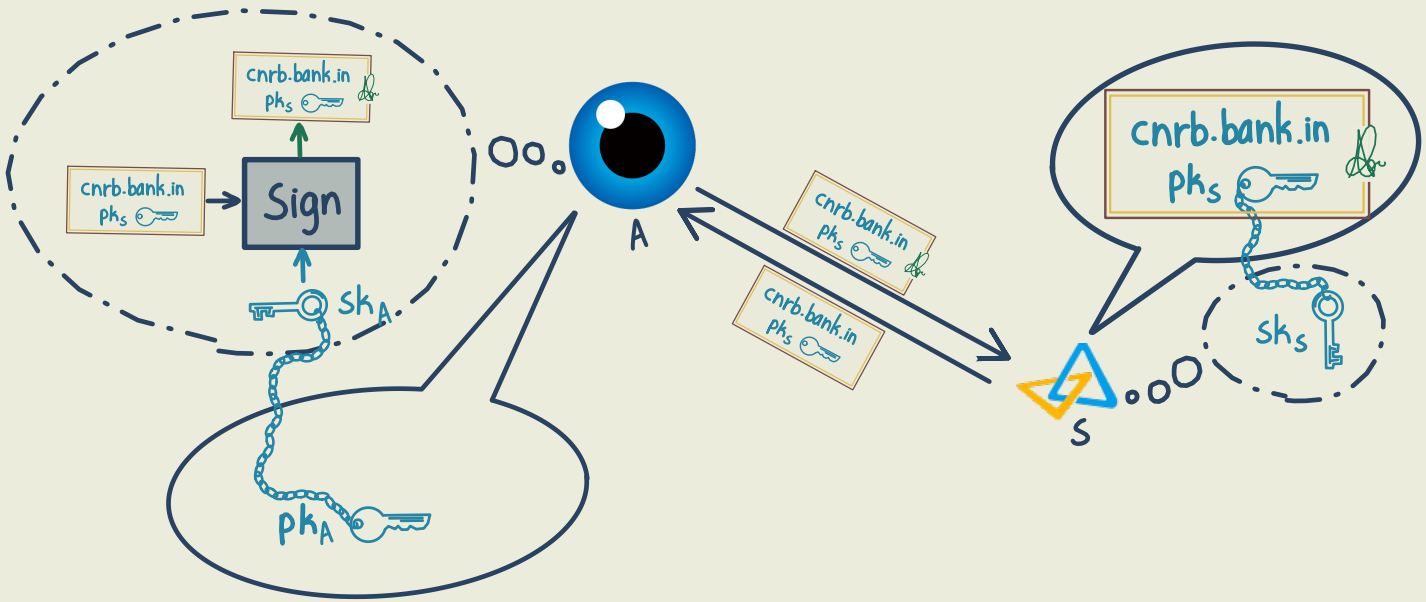
- Using a (public-key) *signing algorithm*

How Exactly does a CA A Issue a Certificate?



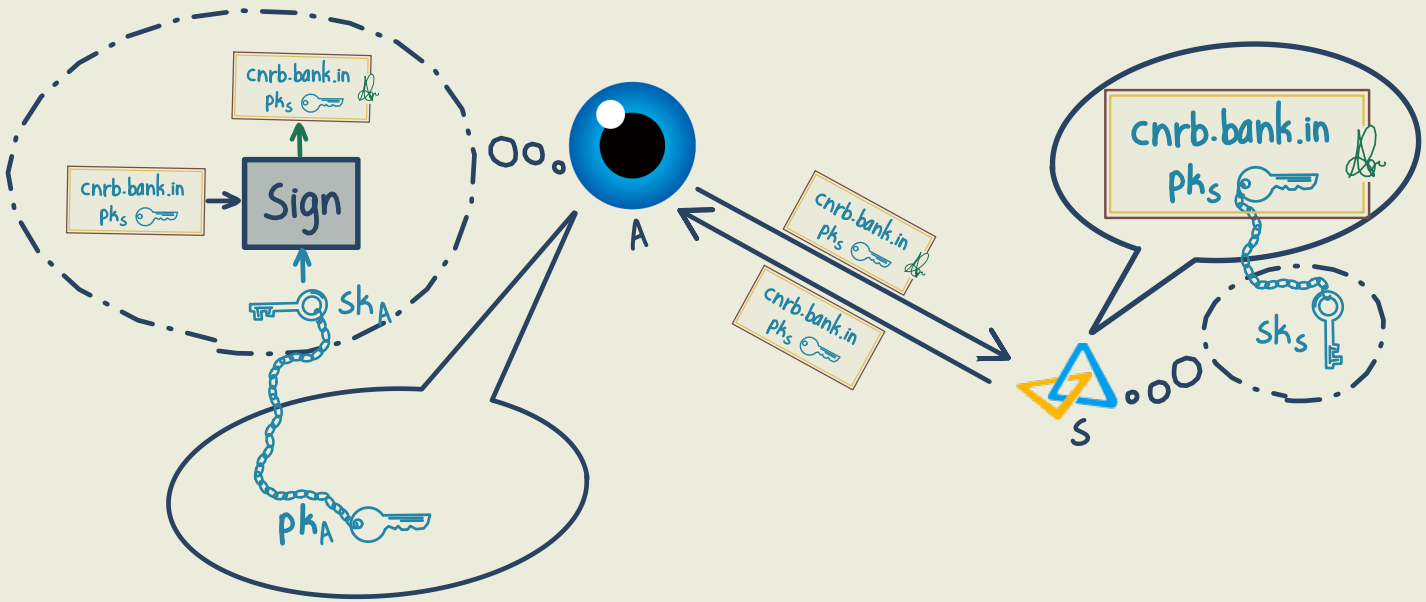
- Using a (public-key) *signing algorithm*
  - A has a public key $\mathtt{pk}_A$ with associated secret key $\mathtt{sk}_A$

How Exactly does a CA *A* Issue a Certificate?



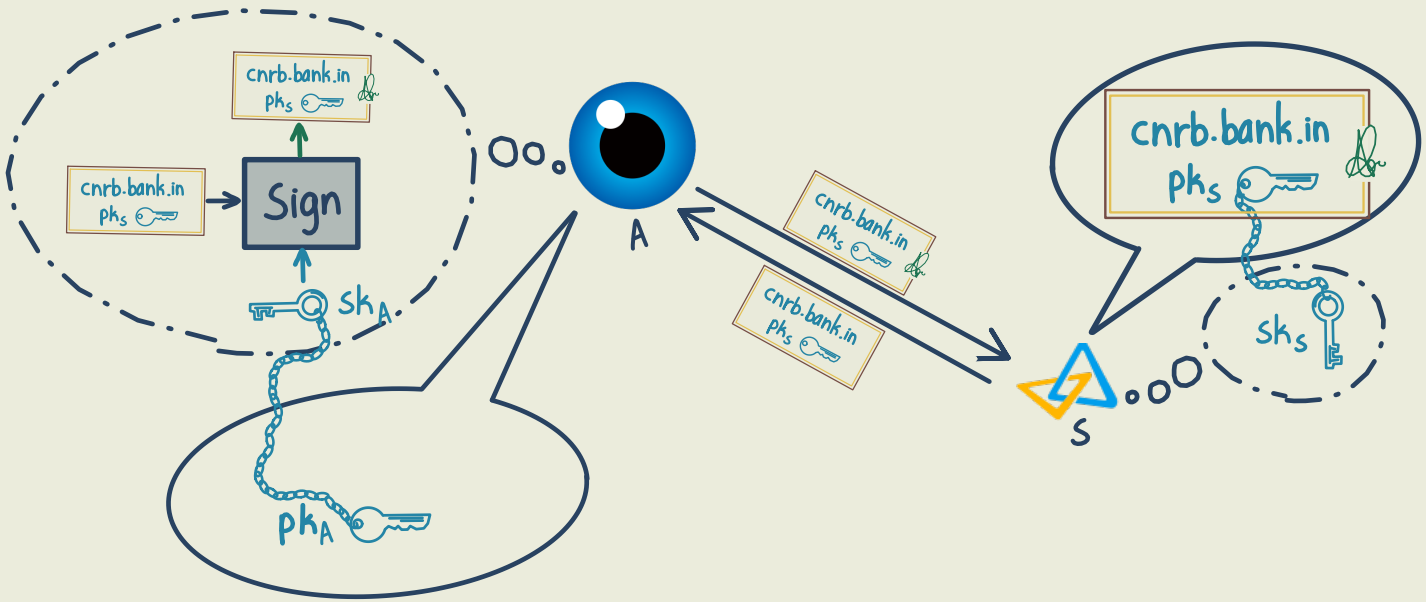
- Using a (public-key) *signing algorithm*
  - *A* has a public key $\mathtt{pk}_A$ with associated secret key $\mathtt{sk}_A$
  - *A* "signs" (the hash of) the certificate *body* under $\mathtt{sk}_A$
  - *Anyone* can use $\mathtt{pk}_A$ to *verify* this signature against the certificate

How Exactly does a CA A Issue a Certificate?



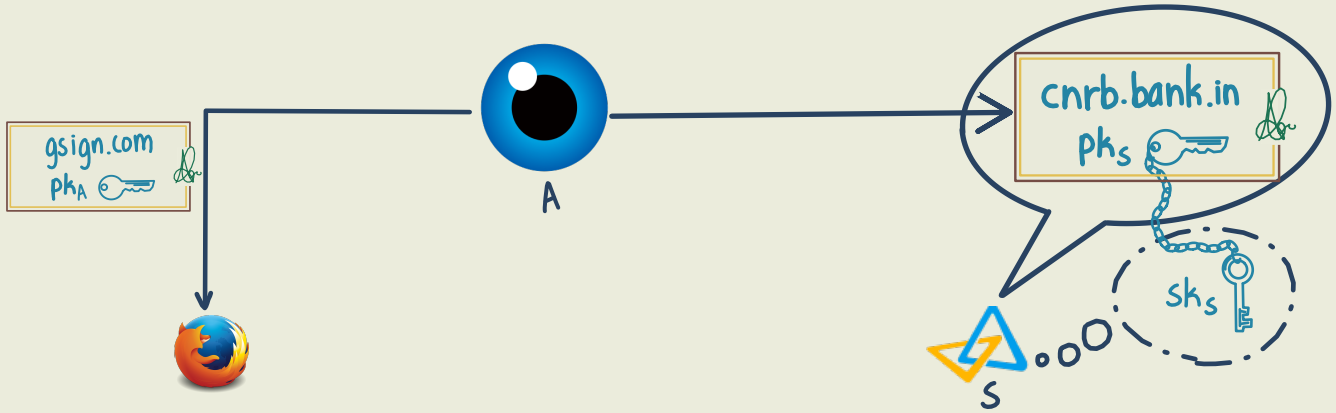
- Using a (public-key) *signing algorithm*
  - A has a public key $\mathtt{pk}_A$ with associated secret key $\mathtt{sk}_A$
  - A "signs" (the hash of) the certificate *body* under $\mathtt{sk}_A$
  - *Anyone* can use $\mathtt{pk}_A$ to *verify* this signature against the certificate
- What are the security requirements?
  - Given only $\mathtt{pk}_A$, no adversary can *forge* new signature that verifies under $\mathtt{pk}_A$

How Exactly does a CA *A* Issue a Certificate?



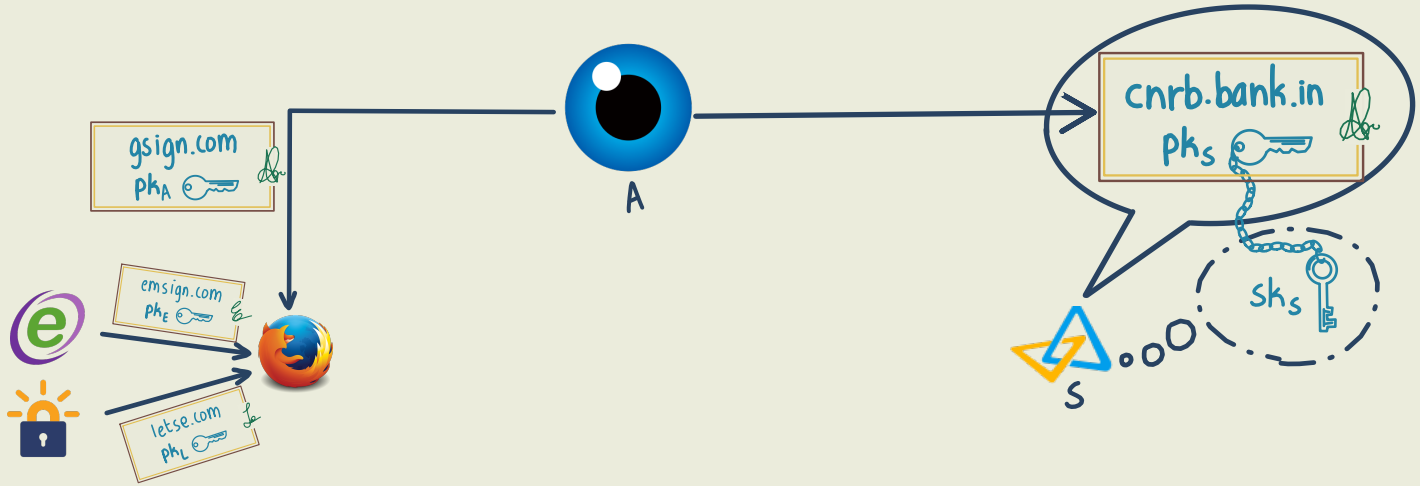
- Using a (public-key) *signing algorithm*
  - *A* has a public key $\mathtt{pk}_A$ with associated secret key $\mathtt{sk}_A$
  - *A* "signs" (the hash of) the certificate *body* under $\mathtt{sk}_A$
  - *Anyone* can use $\mathtt{pk}_A$ to *verify* this signature against the certificate
- What are the security requirements?
  - Given only $\mathtt{pk}_A$, no adversary can *forge* new signature that verifies under $\mathtt{pk}_A$, even having seen prior signatures under $\mathtt{sk}_A$

To Put it All Together..



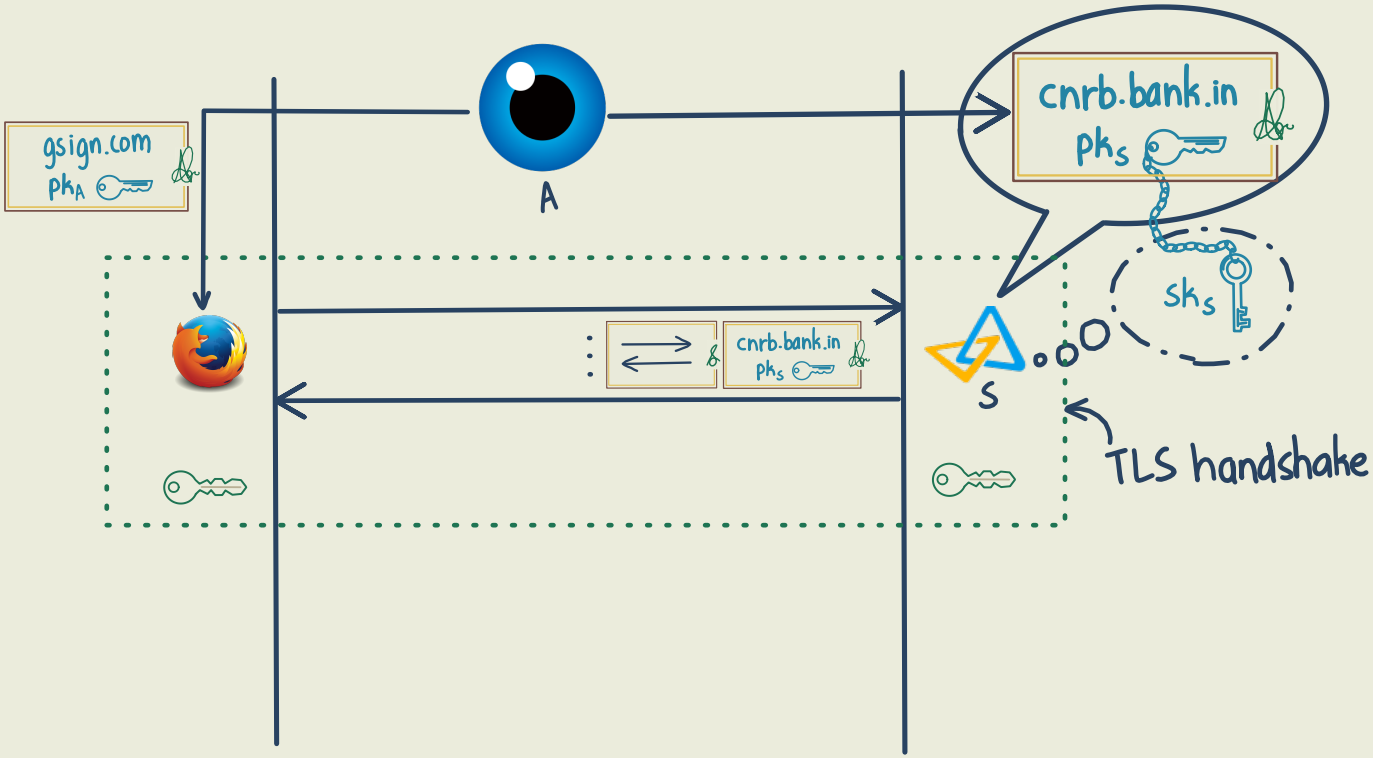
1. shipped with "root certificates", i.e., certificates of trusted CAs

To Put it All Together..



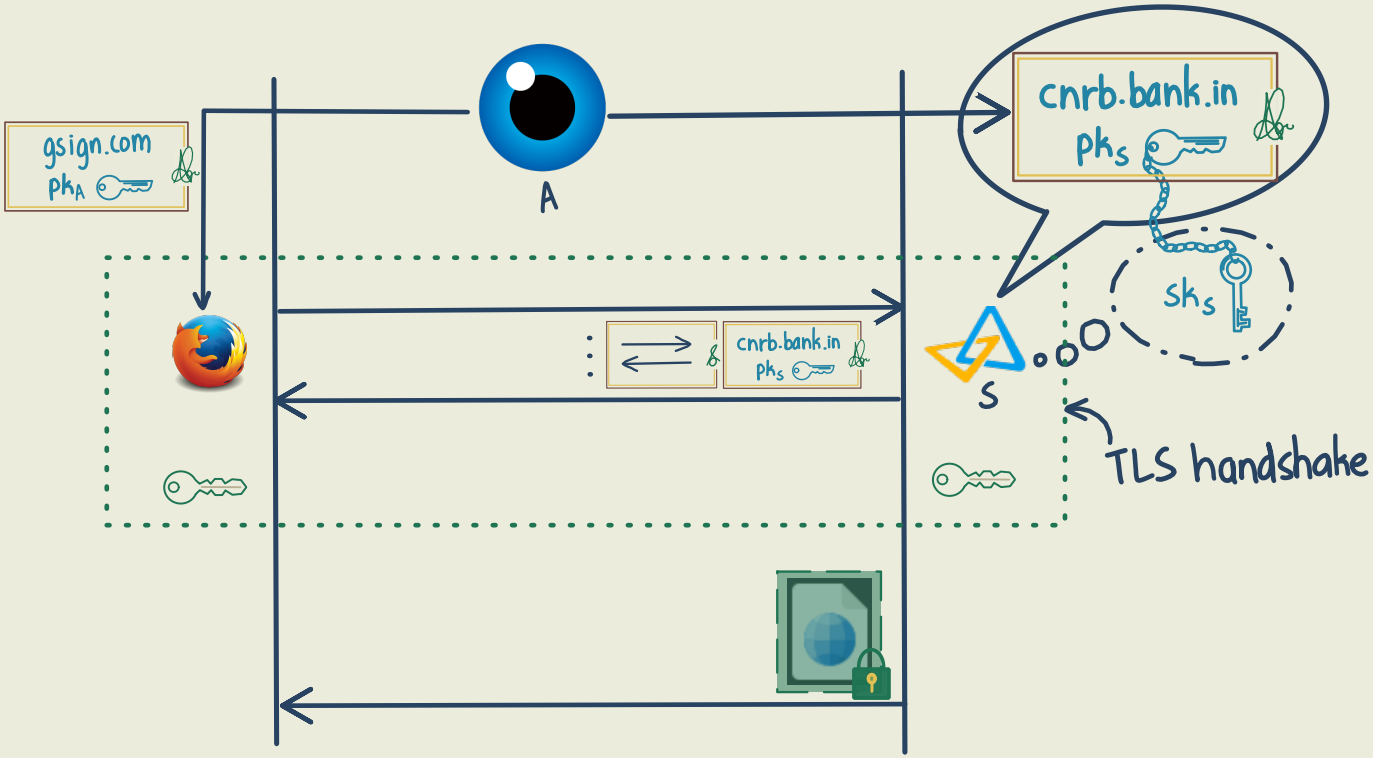
1. shipped with "root certificates", i.e., certificates of trusted CAs

To Put it All Together..



1. shipped with "root certificates", i.e., certificates of trusted CAs
2. When handshakes with , it is provided 's certificate signed by (say) and a signature *on the handshake* under sk_S

To Put it All Together..



1. shipped with "root certificates", i.e., certificates of trusted CAs
2. When handshakes with , it is provided 's certificate signed by (say) and a signature *on the handshake* under sk_S
3. proceeds only if certificate and signature verify

To Put it All Together..



- IRL: there is a “chain” of certificates, with root certificates signing *intermediate* certificates and so on

To Put it All Together..



- IRL: there is a "chain" of certificates, with root certificates signing *intermediate* certificates and so on

For the system to function, we have to trust

1. All CAs, to properly carry out their duties
2. The browser, to not ship root certificates of untrusted CAs

To Put it All Together..

- IRL: there is a "chain" of certificates, with root certificates signing *intermediate* certificates and so on

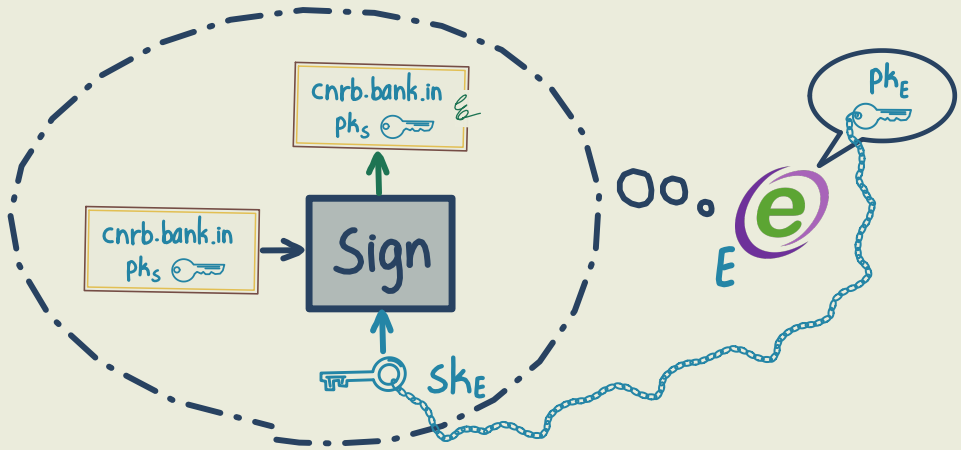
For the system to function, we have to trust

1. All CAs, to properly carry out their duties
2. The browser, to not ship root certificates of untrusted CAs

Our Plan

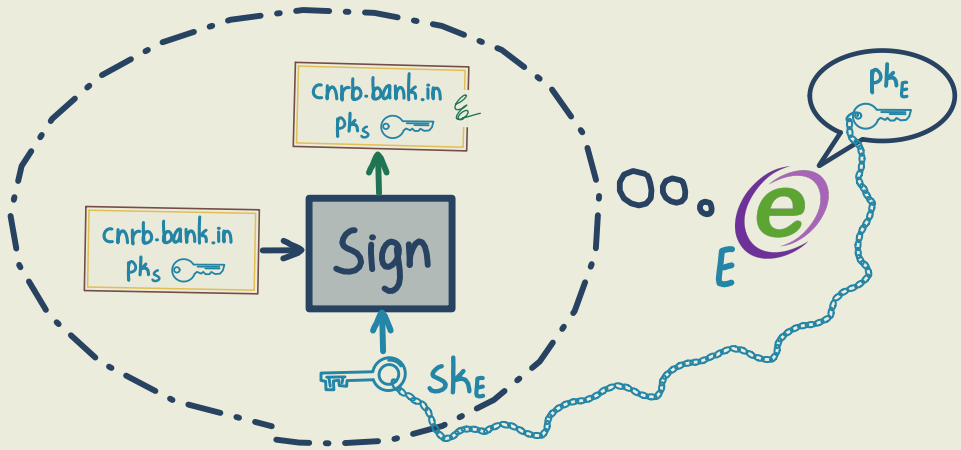
1. Part I: How Identity and Trust Works Over the Internet
2. Part II: What Went Wrong at eMudhra
3. Part III: The Aftermath

...In One Slide..



⚠ Issuing a certificate *pretending to be a CA E* is easy given $\mathtt{sk}_E$

...In One Slide..

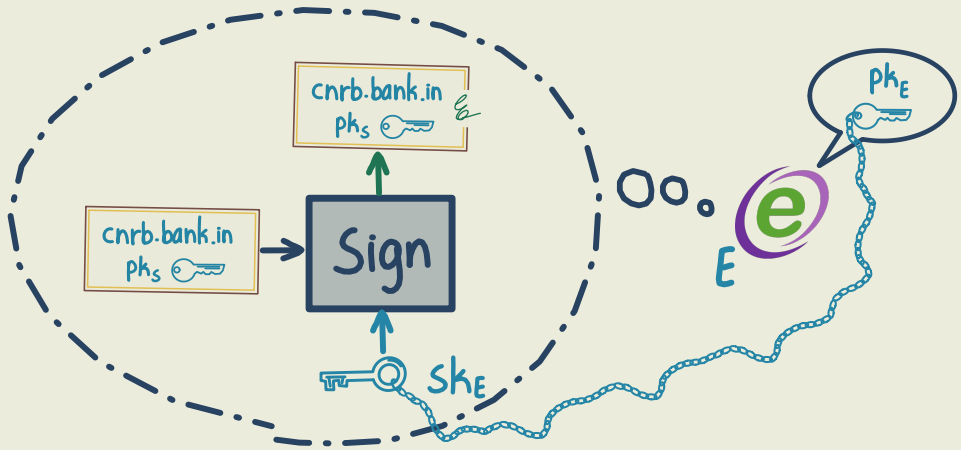


⚠ Issuing a certificate *pretending to be a CA E* is easy given $\mathtt{sk}_E$

☆ Therefore, $\mathtt{sk}_E$ must be:

1. Sampled at random from a very large set of possible keys
   - ⏱ Cost to brute force proportional to the size of this set

...In One Slide..

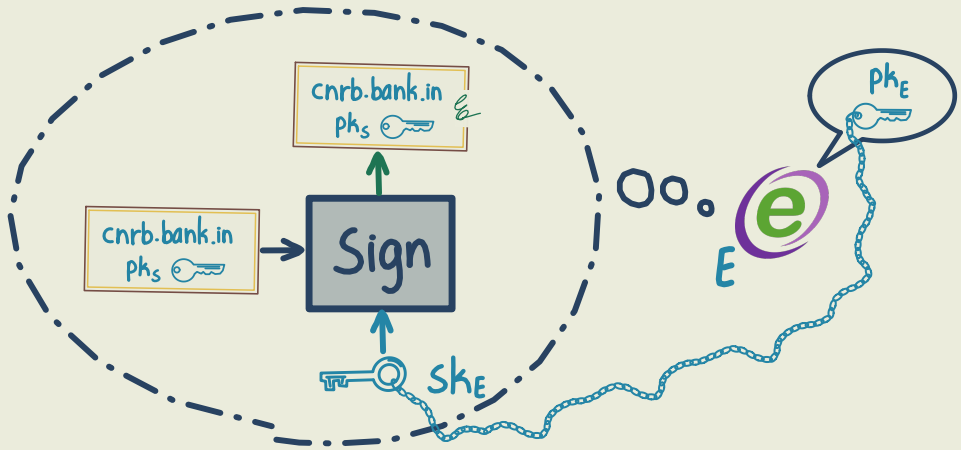


⚠ Issuing a certificate *pretending to be a CA E* is easy given $\mathtt{sk}_E$

☆ Therefore, $\mathtt{sk}_E$ must be:

1. Sampled at random from a very large set of possible keys
   - ⏱ Cost to brute force proportional to the size of this set
2. Hard to compute from the public key $\mathtt{pk}_E$
   - ⏱ For 128 bit security level, cost to compute must be $\approx 2^{128}$

...In One Slide..

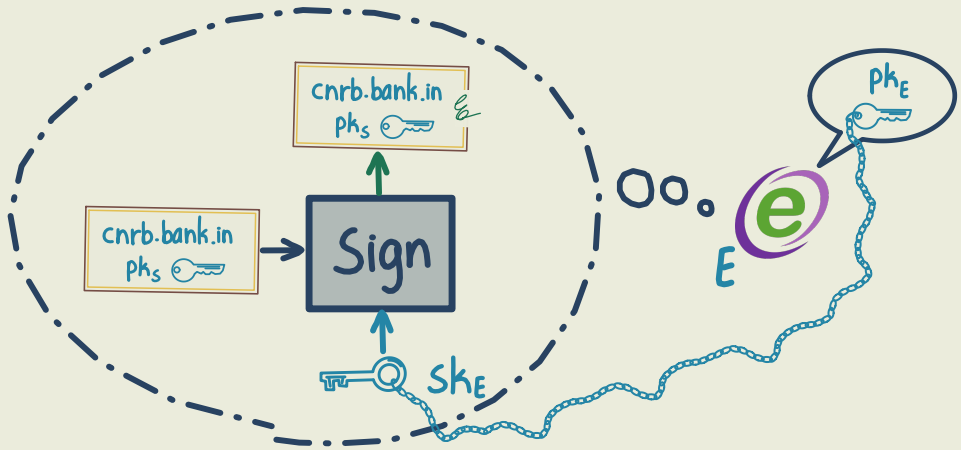


⚠ Issuing a certificate *pretending to be a CA E* is easy given $\mathtt{sk}_E$

☆ Therefore, $\mathtt{sk}_E$ must be:

1. Sampled at random from a very large set of possible keys
   - Cost to brute force proportional to the size of this set
2. Hard to compute from the public key $\mathtt{pk}_E$
   - For 128 bit security level, cost to compute must be $\approx 2^{128}$
   - Belief supported by years of cryptanalysis

...In One Slide..

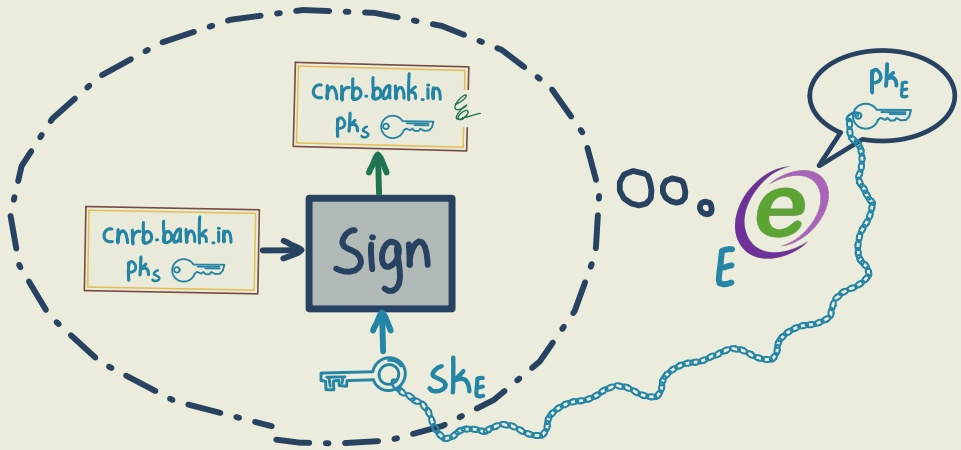


⚠ Issuing a certificate *pretending to be a CA E* is easy given $\mathtt{sk}_E$

☆ Therefore, $\mathtt{sk}_E$ must be:

1. Sampled at random from a very large set of possible keys
   - Cost to brute force proportional to the size of this set
2. Hard to compute from the public key $\mathtt{pk}_E$
   - For 128 bit security level, cost to compute must be $\approx 2^{128}$
   - Belief supported by years of cryptanalysis
3. Securely stored
   - E.g., use an air gap or secure hardware

...In One Slide..



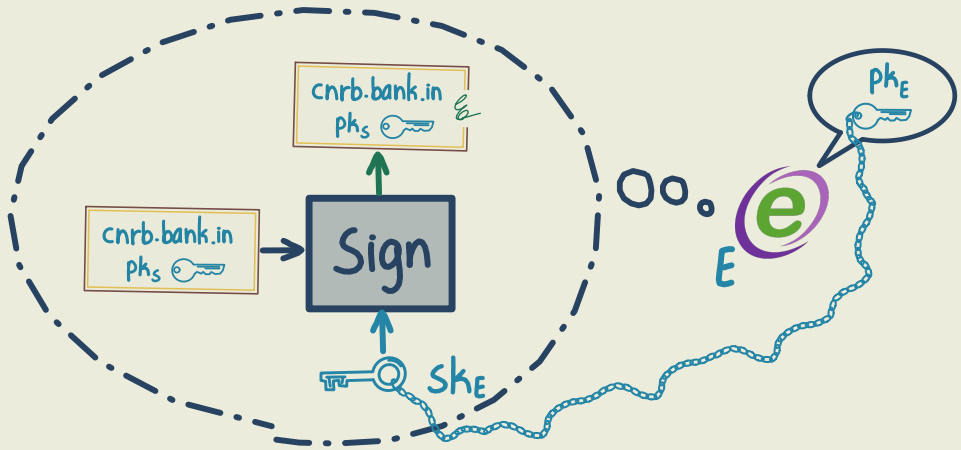
⚠ Issuing a certificate *pretending to be a CA E* is easy given $\mathtt{sk}_E$

☆ Therefore, $\mathtt{sk}_E$ must be:

1. Sampled at random from a very large set of possible keys
   - Cost to brute force proportional to the size of this set
2. Hard to compute from the public key $\mathtt{pk}_E$
   - For 128 bit security level, cost to compute must be $\approx 2^{128}$
   - Belief supported by years of cryptanalysis
3. Securely stored
   - E.g., use an air gap or secure hardware

⚠ What went wrong at eMudhra?

...In One Slide..



⚠ Issuing a certificate *pretending to be a CA E* is easy given $\mathtt{sk}_E$

☆ Therefore, $\mathtt{sk}_E$ must be:

1. Sampled at random from a very large set of possible keys
   - Cost to brute force proportional to the size of this set
2. Hard to compute from the public key $\mathtt{pk}_E$
   - For 128 bit security level, cost to compute must be $\approx 2^{128}$
   - Belief supported by years of cryptanalysis
3. Securely stored
   - E.g., use an air gap or secure hardware

⚠ What went wrong at eMudhra? Secret keys underlying certain compromised (intermediate) certificates were $< 2^8$!

...In One Slide..

⚠ Compromised (intermediate) certificates:

```
EM EV TLS CA - G3B (eMudhra BV), CCADB A012304
SHA-256 E7568FE6E12331F4FD7DE61DA4C7559D0F8EB192A218D4741A3BA75184333838
serial c7c41efe92d825f05eae841a4a82c1, issuer emSign Root TLS CA - G3

EM OV TLS CA - G3B (eMudhra BV), CCADB A012308
SHA-256 98B896625C0C230ADEEC3EB2CF6AAD5001ECF9156A68644E82374CE5833BC25B
serial 6cada367fb1c100088120ea202d568, issuer emSign Root TLS CA - G3

EM DV TLS CA - G3B (eMudhra BV), CCADB A012302
SHA-256 42C4ACCD12C4F59597925E1A08A7A901026085438093O8D50EC427E118BA1493
serial a4aee678f23362b4c0e460b772f214, issuer emSign Root TLS CA - G3

EM EV TLS CA - G3C (eMudhra Inc), CCADB A012296
SHA-256 432EC25F0D28DE9006D97BF752F17BBB2CD41F669933FCB072D9A1148D0261F3
serial 8009928a64107601af05fa2f19f109, issuer emSign Root TLS CA - G3

EM OV TLS CA - G3C (eMudhra Inc), CCADB A012300
SHA-256 CB6E15165AC802E9B3F993F16C25D41F4E81C21EBDE71AE1B012AC00BF9C7C3F
serial 296926c227a3a1526392b19482a504, issuer emSign Root TLS CA - G3

EM DV TLS CA - G3C (eMudhra Inc), CCADB A012294
SHA-256 0164D7CEAE5BB909C272AF151FF44B1177AE9B47D9424B94F1D9558C30934D19
serial 588b2e3c2bb8cda13e1a1209254321, issuer emSign Root TLS CA - G3
```

©bugzilla.mozilla.org

...In One Slide..

⚠ Compromised (intermediate) certificates:

```
EM EV TLS CA - G3B (eMudhra BV), CCADB A012304
SHA-256 E7568FE6E12331F4FD7DE61DA4C7559D0F8EB192A218D4741A3BA75184333838
serial c7c41efe92d825f05eae841a4a82c1, issuer emSign Root TLS CA - G3

EM OV TLS CA - G3B (eMudhra BV), CCADB A012308
SHA-256 98B896625C0C230ADEEC3EB2CF6AAD5001ECF9156A68644E82374CE5833BC25B
serial 6cada367fb1c100088120ea202d568, issuer emSign Root TLS CA - G3

EM DV TLS CA - G3B (eMudhra BV), CCADB A012302
SHA-256 42C4ACCD12C4F59597925E1A08A7A90102608543809308D50EC427E118BA1493
serial a4aee678f23362b4c0e460b772f214, issuer emSign Root TLS CA - G3

EM EV TLS CA - G3C (eMudhra Inc), CCADB A012296
SHA-256 432EC25F0D28DE9006D97BF752F17BBB2CD41F669933FCB072D9A1148D0261F3
serial 8009928a64107601af05fa2f19f109, issuer emSign Root TLS CA - G3

EM OV TLS CA - G3C (eMudhra Inc), CCADB A012300
SHA-256 CB6E15165AC802E9B3F993F16C25D41F4E81C21EBDE71AE1B012AC00BF9C7C3F
serial 296926c227a3a1526392b19482a504, issuer emSign Root TLS CA - G3

EM DV TLS CA - G3C (eMudhra Inc), CCADB A012294
SHA-256 0164D7CEAE5BB909C272AF151FF44B1177AE9B47D9424B94F1D9558C30934D19
serial 588b2e3c2bb8cda13e1a1209254321, issuer emSign Root TLS CA - G3
```

©bugzilla.mozilla.org

INTERMEDIATE	TYPE	D
EM DV TLS CA - G3B	TLS	156
EM DV TLS CA - G3C	TLS	230
EM EV TLS CA - G3B	TLS	115
EM EV TLS CA - G3C	TLS	56
EM OV TLS CA - G3B	TLS	155
EM OV TLS CA - G3C	TLS	4
EM OV CS CA - G3B	Code signing	66
EM OV CS CA - G3C	Code signing	237

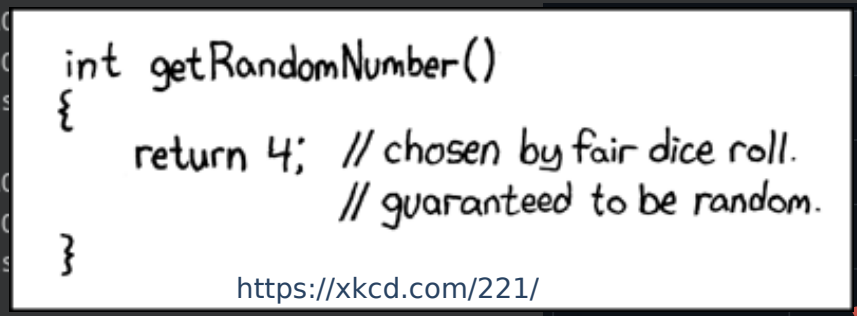
©scotthelme.co.uk

...In One Slide..

⚠ Compromised (intermediate) certificates:

```
EM EV TLS CA - G3B (eMudhra BV), CCADB A012304
SHA-256 E7568FE6E12331F4FD7DE61DA4C7559D0F8EB192A218D4741A3BA75184333838
serial c7c41efe92d825f05eae841a4a82c1, issuer emSign Root TLS CA - G3

EM OV TLS CA - G3B (eMudhra BV), CCADB A
SHA-256 98B896625C0C230ADEEC3EB2CF6AAD50
serial 6cada367fb1c100088120ea202d568, is

EM DV TLS CA - G3B (eMudhra BV), CCADB A
SHA-256 42C4ACCD12C4F59597925E1A08A7A901
serial a4aee678f23362b4c0e460b772f214, is

EM EV TLS CA - G3C (eMudhra Inc), CCADB A012296
SHA-256 432EC25F0D28DE9006D97BF752F17BBB2CD41F669933FCB072D9A1148D0261F3
serial 8009928a64107601af05fa2f19f109, issuer emSign Root TLS CA - G3

EM OV TLS CA - G3C (eMudhra Inc), CCADB A012300
SHA-256 CB6E15165AC802E9B3F993F16C25D41F4E81C21EBDE71AE1B012AC00BF9C7C3F
serial 296926c227a3a1526392b19482a504, issuer emSign Root TLS CA - G3

EM DV TLS CA - G3C (eMudhra Inc), CCADB A012294
SHA-256 0164D7CEAE5BB909C272AF151FF44B1177AE9B47D9424B94F1D9558C30934D19
serial 588b2e3c2bb8cda13e1a1209254321, issuer emSign Root TLS CA - G3
```

©bugzilla.mozilla.org



https://xkcd.com/221/

		D
		156
		230
		115
		56
EM OV TLS CA - G3B	TLS	155
EM OV TLS CA - G3C	TLS	4
EM OV CS CA - G3B	Code signing	66
EM OV CS CA - G3C	Code signing	237

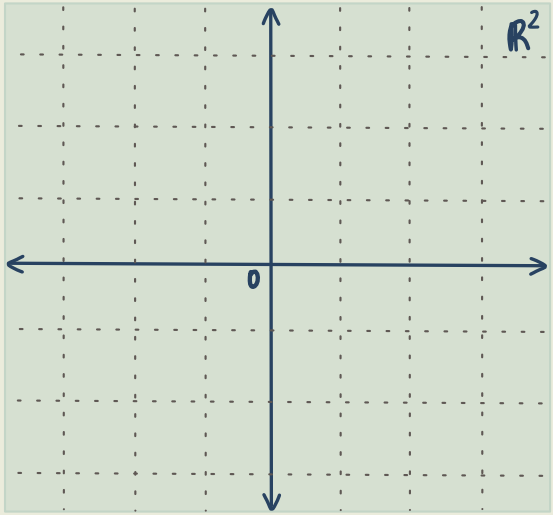
©scotthelme.co.uk

...In Three Slides...

- eMudhra used ECDSA: elliptic-curve (digital) signing algorithm

...In Three Slides...

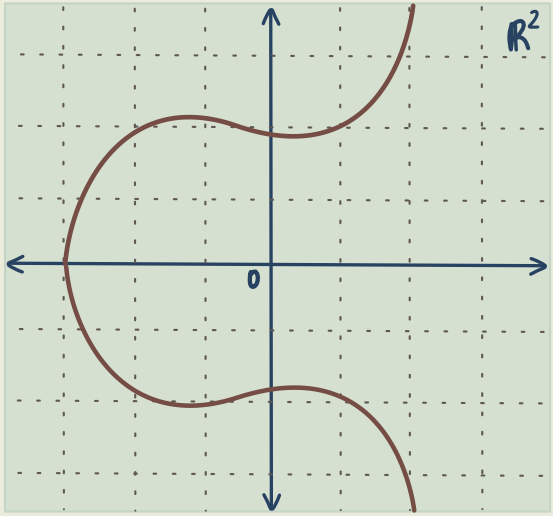
- eMudhra used ECDSA: elliptic-curve (digital) signing algorithm



- What is an elliptic curve (after oversimplifying)?

...In Three Slides...

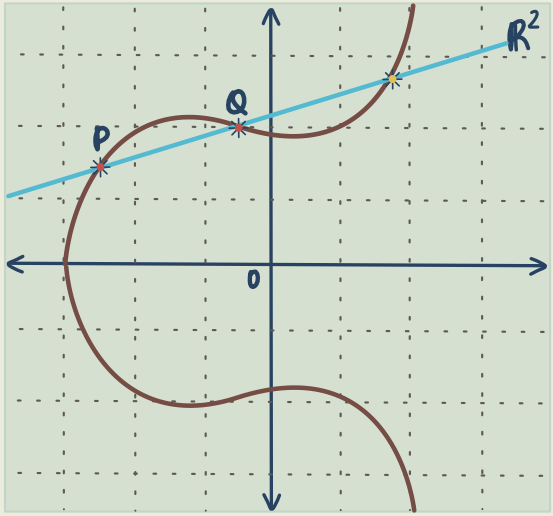
- eMudhra used ECDSA: elliptic-curve (digital) signing algorithm



- What is an elliptic curve (after oversimplifying)?
  - A set of points (x, y) such that $y^2 = x^3 + ax + b \bmod p$

...In Three Slides...

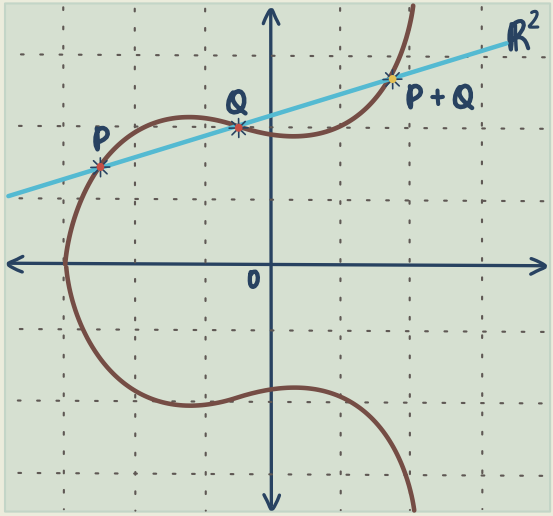
- eMudhra used ECDSA: elliptic-curve (digital) signing algorithm



- What is an elliptic curve (after oversimplifying)?
  - A set of points (x, y) such that $y^2 = x^3 + ax + b \bmod p$
    - Fact 1: It is easy to compute the number of points ℓ on the curve
    - Fact 2: Every line intersecting the curve at two points, must intersect it at a third point*

...In Three Slides...

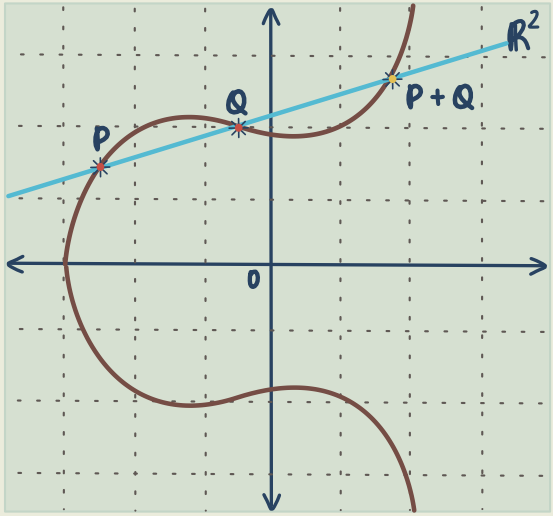
- eMudhra used ECDSA: elliptic-curve (digital) signing algorithm



- What is an elliptic curve (after oversimplifying)?
  - A set of points (x, y) such that $y^2 = x^3 + ax + b \bmod p$
    - Fact 1: It is easy to compute the number of points ℓ on the curve
    - Fact 2: Every line intersecting the curve at two points, must intersect it at a third point*
  - Along with a binary operation $+$
    - $P + Q$ defined as the third point on line passing through P and Q
    - In particular $P + P = 2P$ defined using the tangent at P

...In Three Slides...

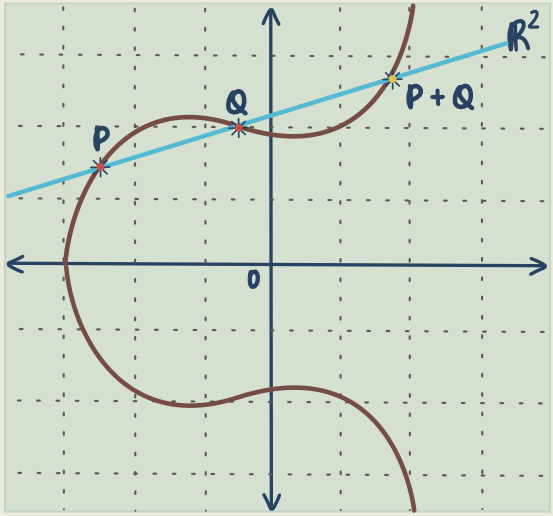
- eMudhra used ECDSA: elliptic-curve (digital) signing algorithm



- What is an elliptic curve (after oversimplifying)?
  - A set of points (x, y) such that $y^2 = x^3 + ax + b \bmod p$
    - Fact 1: It is easy to compute the number of points ℓ on the curve
    - Fact 2: Every line intersecting the curve at two points, must intersect it at a third point*
  - Along with a binary operation $+$
    - $P + Q$ defined as the third point on line passing through P and Q
    - In particular $P + P = 2P$ defined using the tangent at P
    - Fact 3: for a (random) base point P on the curve, the walk $P \to 2P \to 3P \to \ldots \to \ell P$ will cover the whole curve

...In Three Slides...

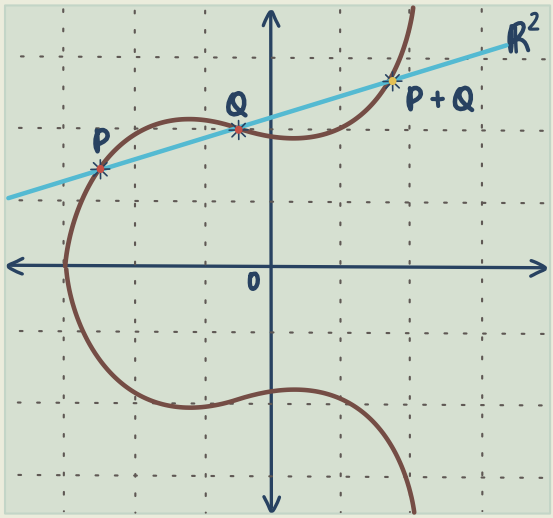
- ...



- eMudhra used NIST P-384 elliptic curve:
  - $p := 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
  - $a = -3$
  - $b := 27580193559959705877849...65439761230575$ (116 digits)
  - $\ell := 39402006196394479212279...54433653942643$ (116 digits)

...In Three Slides...

- ...



- eMudhra used NIST P-384 elliptic curve:
  - $p := 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
  - $a = -3$
  - $b := 27580193559959705877849...65439761230575$ (116 digits)
  - $\ell := 39402006196394479212279...54433653942643$ (116 digits)
- ECDSA key-pair is of the form:
  - Secret key $\mathtt{sk}_E$: random walk-length from $\{1, \ldots, \ell\}$
  - Public key $\mathtt{pk}_E$: $(P, \mathtt{sk}P)$ for a random base point P

...In Three Slides...



How is random $\texttt{sk}_E \in \{1, \ldots, \ell\}$ usually sampled?

- Software-based random-number generators (RNGs)
- Hardware-assisted RNGs

...In Three Slides...



How is random $\texttt{sk}_E \in \{1, \ldots, \ell\}$ usually sampled?

- Software-based random-number generators (RNGs)
- Hardware-assisted RNGs



In eMudhra's case compromised certificates had $\texttt{sk}_E < 2^8$ (a byte)

...In Three Slides...



How is random $\texttt{sk}_E \in \{1, \ldots, \ell\}$ usually sampled?

- Software-based random-number generators (RNGs)
- Hardware-assisted RNGs



In eMudhra's case compromised certificates had $\texttt{sk}_E < 2^8$ (a byte)

- eMudhra claims:

> The matter concerns 8 of eMudhra's several intermediate CAs that were trusted as part of its Global TLS business. These issuing CAs were created under eMudhra's ECC root in a Key Generation Ceremony (KGC) held on April 30, 2025, using FIPS 140-2 Level 3 Hardware Security Modules (HSMs).
>
> A flaw in the KGC caused a wrap/unwrap step to produce a second set of weak keys alongside the original strong keys. The weak keys were the ones published to the browser root stores. eMudhra is working to establish the root cause.

...In Three Slides...



How is random $\texttt{sk}_E \in \{1, \ldots, \ell\}$ usually sampled?

- Software-based random-number generators (RNGs)
- Hardware-assisted RNGs



In eMudhra's case compromised certificates had $\texttt{sk}_E < 2^8$ (a byte)

- eMudhra claims:

> The matter concerns 8 of eMudhra's several intermediate CAs that were trusted as part of its Global TLS business. These issuing CAs were created under eMudhra's ECC root in a Key Generation Ceremony (KGC) held on April 30, 2025, using FIPS 140-2 Level 3 Hardware Security Modules (HSMs).
>
> A flaw in the KGC caused a wrap/unwrap step to produce a second set of weak keys alongside the original strong keys. The weak keys were the ones published to the browser root stores. eMudhra is working to establish the root cause.



What exactly was the flaw?

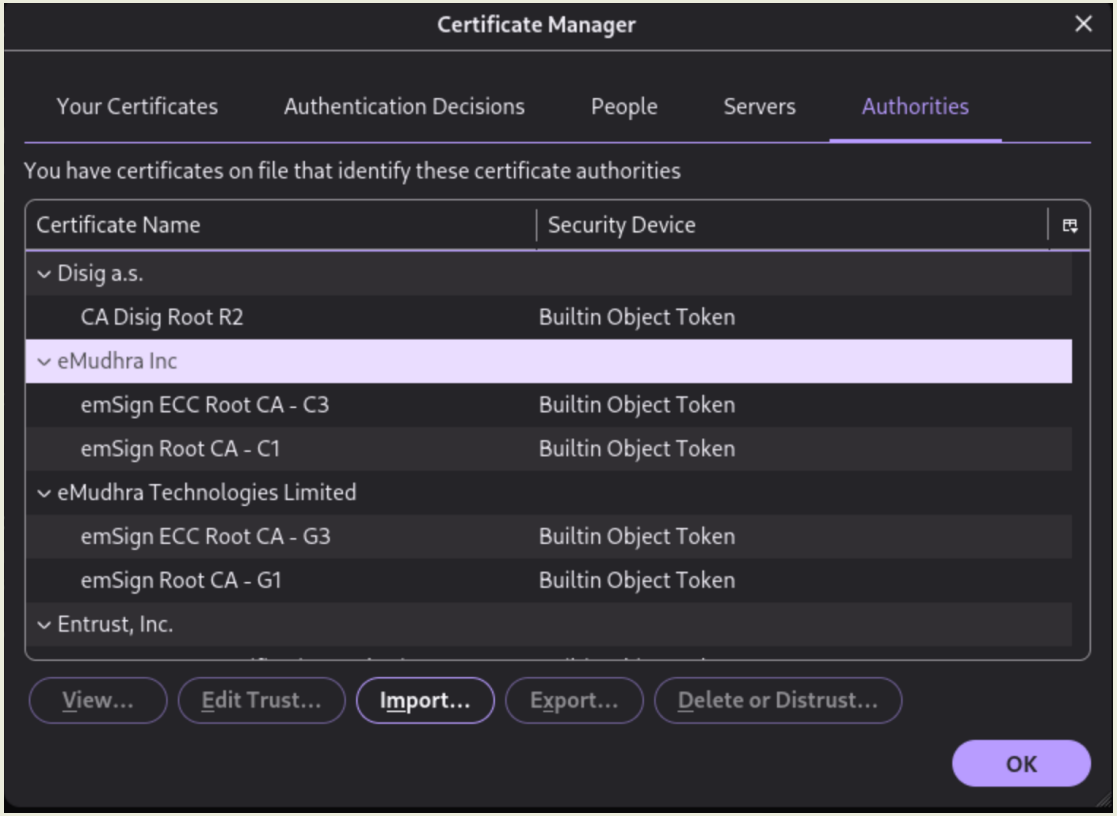


How did the security researcher discover this?

Our Plan

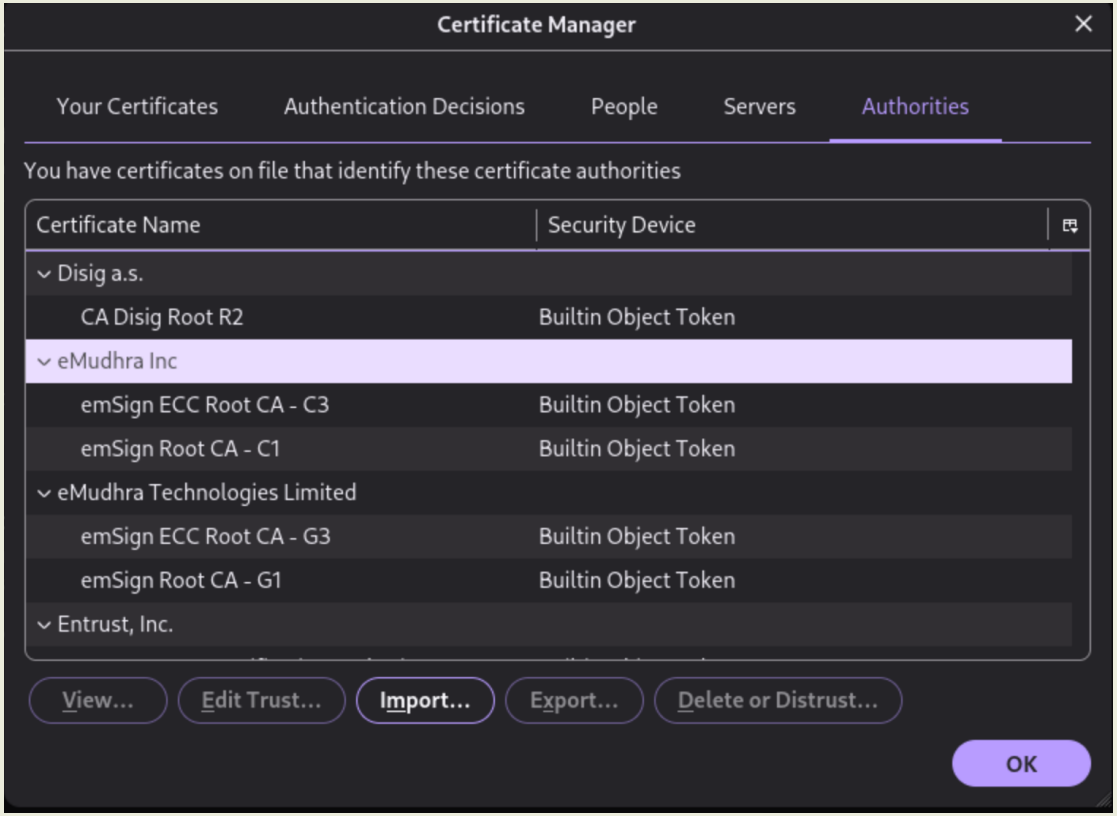
1. Part I: How Identity and Trust Works Over the Internet
2. Part II: What Went Wrong at eMudhra
3. Part III: The Aftermath

Why Did Browsers Scramble to Ship Updates?



- To “revoke” compromised eMudhra certificates (add to CRLset)

Why Did Browsers Scramble to Ship Updates?



- To “revoke” compromised eMudhra certificates (add to CRLset)
- An audit of Certificate Transparency ledger indicated that the compromised certificates were “not used” during the compromise window (Apr/2025-)

How were/are Developer Tools Affected?



Apple's git still trusts the compromised eMudhra certificates

28th October 2026

Apple was notified of the eMudhra incident on 17th October. As of 28th October, the developer tools Apple ships with macOS, including git, curl and python3, still trust the compromised certificates.

Worse, macOS 27.2, released on 26th October, did not fix the issue and undid the workaround that users had applied by hand, leaving them vulnerable again.

emudra.we-have-a-problem.com

- eMudhra root certificate shipped (e.g.) with MacOS
  - Used by git, curl or python3 on MacOS

How were/are Developer Tools Affected?



Apple's git still trusts the compromised eMudhra certificates

28th October 2026

Apple was notified of the eMudhra incident on 17th October. As of 28th October, the developer tools Apple ships with macOS, including git, curl and python3, still trust the compromised certificates.

Worse, macOS 27.2, released on 26th October, did not fix the issue and undid the workaround that users had applied by hand, leaving them vulnerable again.

emudra.we-have-a-problem.com

- eMudhra root certificate shipped (e.g.) with MacOS
  - Used by git, curl or python3 on MacOS

⚠ Apple still hasn't yet revoked the intermediate certificates!

Why IoT Devices are Sitting Ducks

- **IoT, embedded and OT devices.**
  Cameras, routers, cars, medical devices, smart appliances, payment terminals and industrial controllers phoning home for commands and updates. Their bundles are frozen until 2039, which is the "cannot be contained" point made real.

©scotthelme.co.uk

⚠ Certificate bundles frozen till 2039!

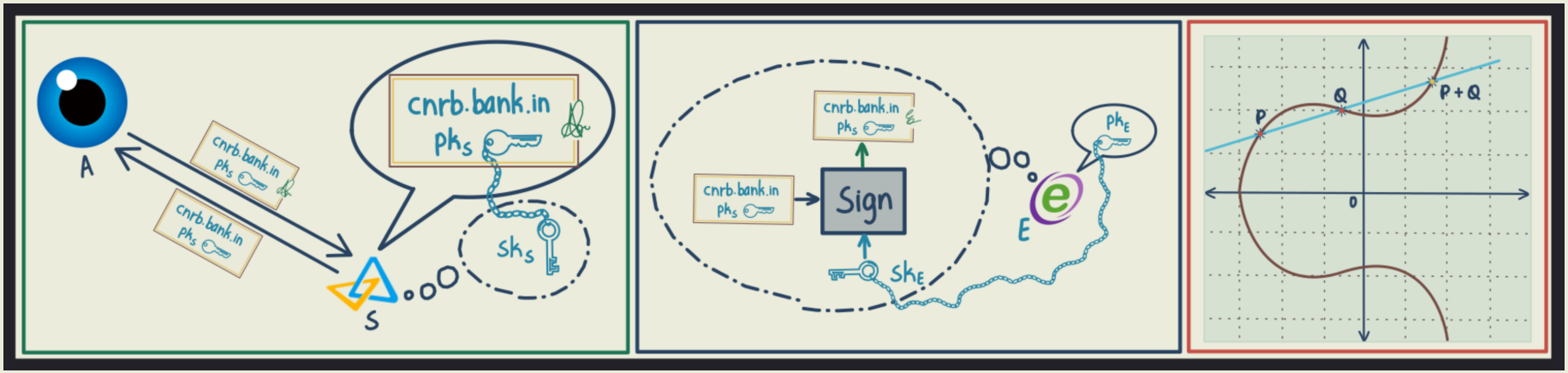
Why IoT Devices are Sitting Ducks

- **IoT, embedded and OT devices.**
  Cameras, routers, cars, medical devices, smart appliances, payment terminals and industrial controllers phoning home for commands and updates. Their bundles are frozen until 2039, which is the "cannot be contained" point made real.

©scotthelme.co.uk

⚠ Certificate bundles frozen till 2039!

- And there is more...
  - Windows code signing
  - CI/CD pipelines
  - ...



What Really... Happened at eMudhra?

Chethan Kamath

Further Material

1. The compromise was disclosed at *emudhra.we-have-a-problem.com*
2. Bugzilla tracker for the compromise
3. Relevant blogposts/news articles:
   - arsTECHNICA article
   - Firstpost article
   - scotthelme.co.uk: *Eight Bits of Trust; eMudhra's CA keys cracked in milliseconds*
   - johncook.com: Elliptic Curve P-384
4. eMudhra press release can be found here