

# Web site security: authentication, encryption, and public-key certificates

Designing Secure Systems

Mark D. Ryan and David Galindo  
University of Birmingham

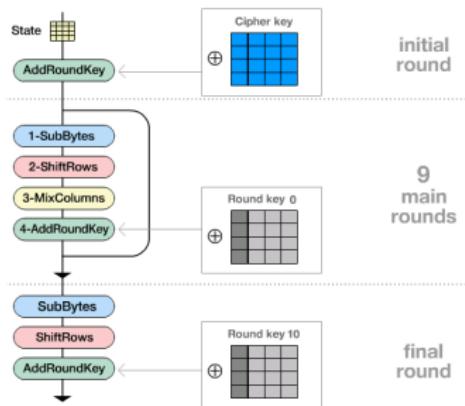
# Contents

- 1) Cryptography
- 2) Authenticating websites and encrypting web traffic
- 3) The Certificate Authority model and its problems
  - The currently-used system
- 4) Certificate transparency
  - The being-rolled-out system
- 5) Certificate revocation

# 1. Cryptography

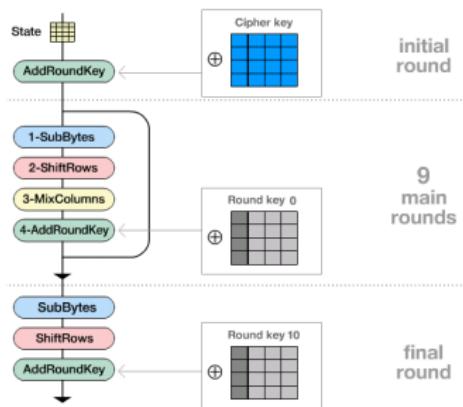
# Symmetric key encryption (e.g., AES)

AES block cipher (a permutation substitution network)

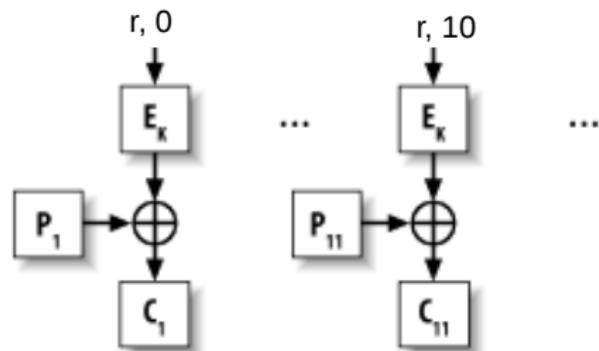


# Symmetric key encryption (e.g., AES)

AES block cipher (a permutation substitution network)

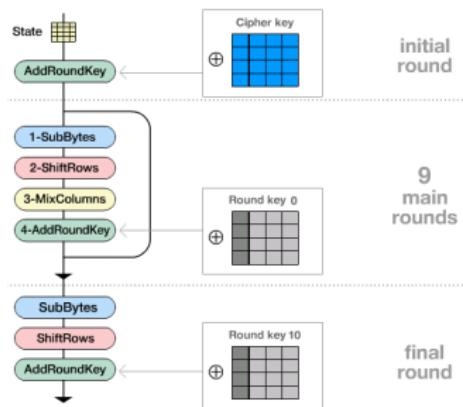


Counter-mode (a mode of operation for block ciphers)

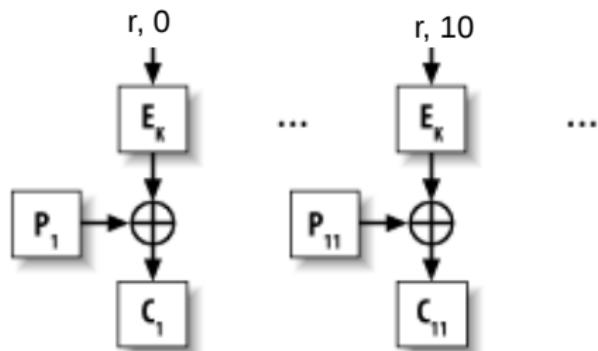


# Symmetric key encryption (e.g., AES)

AES block cipher (a permutation substitution network)



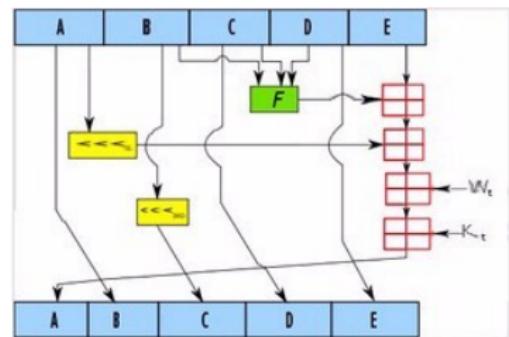
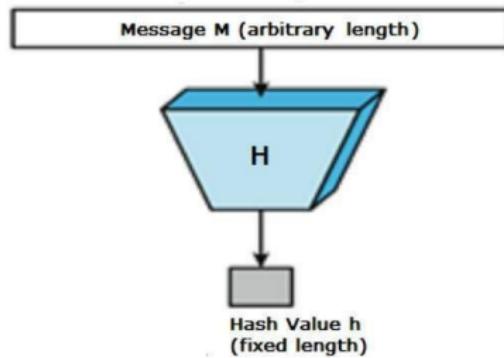
Counter-mode (a mode of operation for block ciphers)



$$\text{dec}(k, r, \text{enc}(k, r, m)) = m$$

$$\text{dec}(k, \text{enc}(k, m)) = m$$

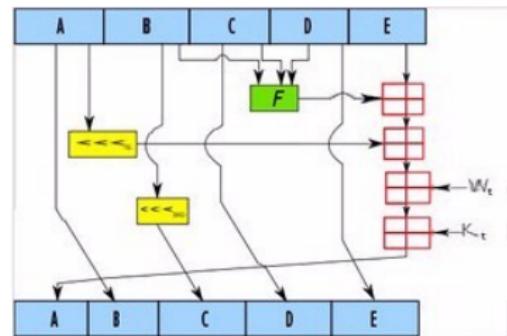
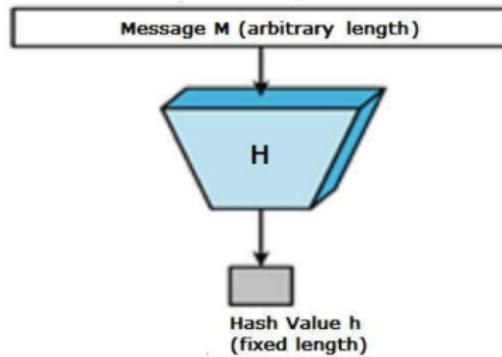
# Hash functions (e.g., SHA2)



$H : \{\text{messages of any size}\} \rightarrow \{256\text{-bit messages}\}$   
such that:

$H(x) = H(y)$  implies  $x = y$  with huge probability

# Hash functions (e.g., SHA2)



$H : \{\text{messages of any size}\} \rightarrow \{256\text{-bit messages}\}$   
such that:

$H(x) = H(y)$  implies  $x = y$  with huge probability

# Public-key encryption (e.g., RSA-OAEP)

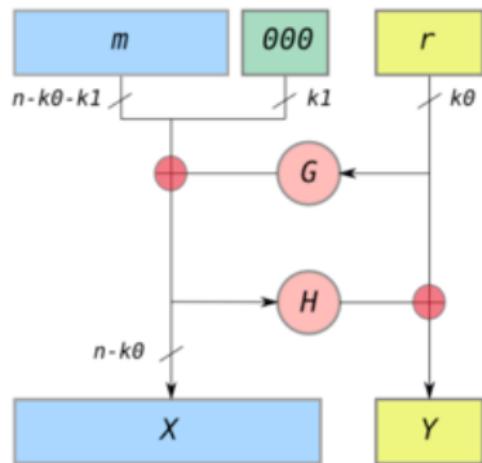
$p, q$  prime

$$n = p * q$$

$$e * d = 1 \bmod (p-1)(q-1)$$

$$C = M^e \bmod n$$

$$M = C^d \bmod n$$



$$\text{dec}(\text{dk}(k), \text{enc}(\text{ek}(k), r, m)) = m$$

# Hybrid encryption

- Unlike the case for symmetric key encryption, there are no “block modes” for public key encryption
  - Because public key encryption is too slow to do for very long messages.
- To encrypt a long message  $m$  using public key encryption with a public key  $pk$ :
  - Choose a random symmetric key,  $k$
  - Encrypt  $m$  with  $k$ :  $c1=enc(k, r1, m)$
  - Encrypt  $k$  with  $pk$ :  $c2=enc(pk, r2, k)$
- Send  $c1, r1, c2$ .

# Hybrid encryption

- Unlike the case for symmetric key encryption, there are no “block modes” for public key encryption
  - Because public key encryption is too slow to do for very long messages.
- To encrypt a long message  $m$  using public key encryption with a public key  $pk$ :
  - Choose a random symmetric key,  $k$
  - Encrypt  $m$  with  $k$ :  $c1=enc(k, r1, m)$
  - Encrypt  $k$  with  $pk$ :  $c2=enc(pk, r2, k)$
  - Send  $c1, r1, c2$ .

# Digital signatures (e.g., Schnorr)

- Initialization (security parameter  $k$ )  
 $p, q$ , two large primes such that

$$\begin{aligned}q &\mid (p-1) \\2^{k-1} &\leq q < 2^k\end{aligned}$$

$g$ , element of  $\mathbb{Z}_p^*$  of order  $q$

$f$ , hash function

secret key  $x \in \mathbb{Z}_q^*$

public key  $y = g^{-x} \bmod p$

- Signature

- $- K \in \mathbb{Z}_q^*$
- $- r = g^K \bmod p$
- $- e = f(m, r)$
- $- s = K + xe \bmod q$
- $- \sigma_1 = r$  and  $\sigma_2 = s$

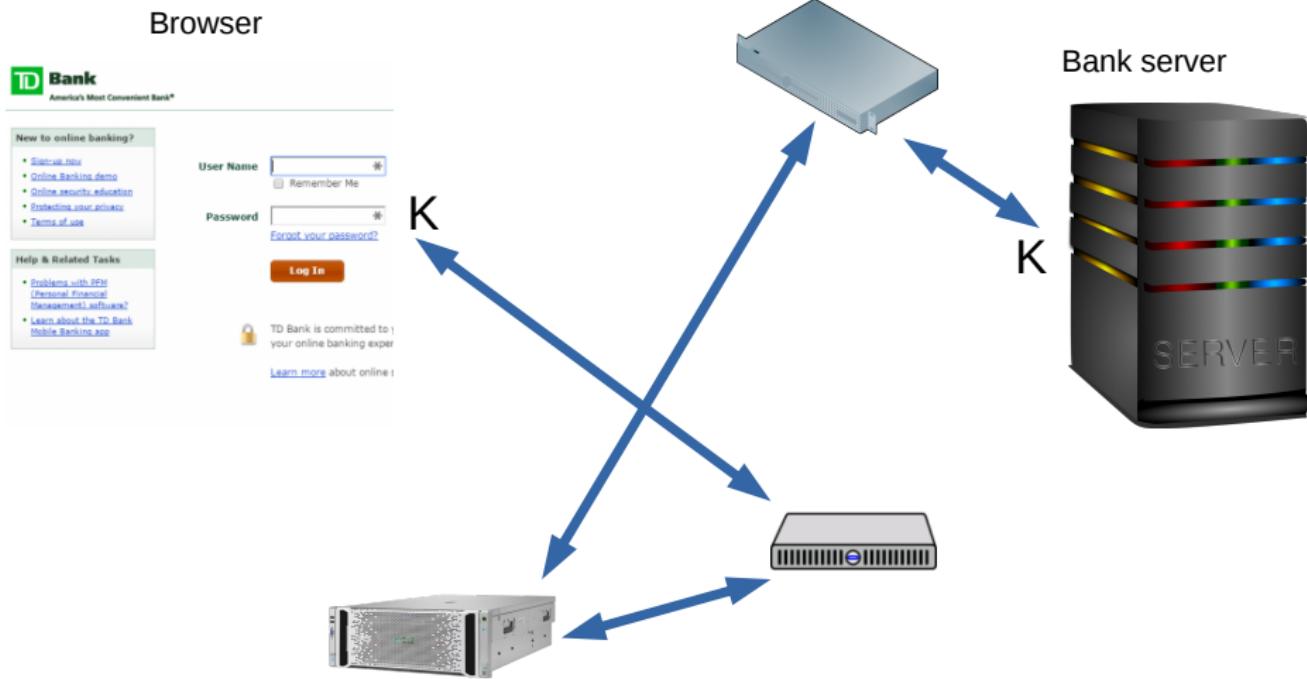
- Verification

- $- e \stackrel{?}{=} f(m, r)$
- $- r \stackrel{?}{=} g^s y^e \bmod p$

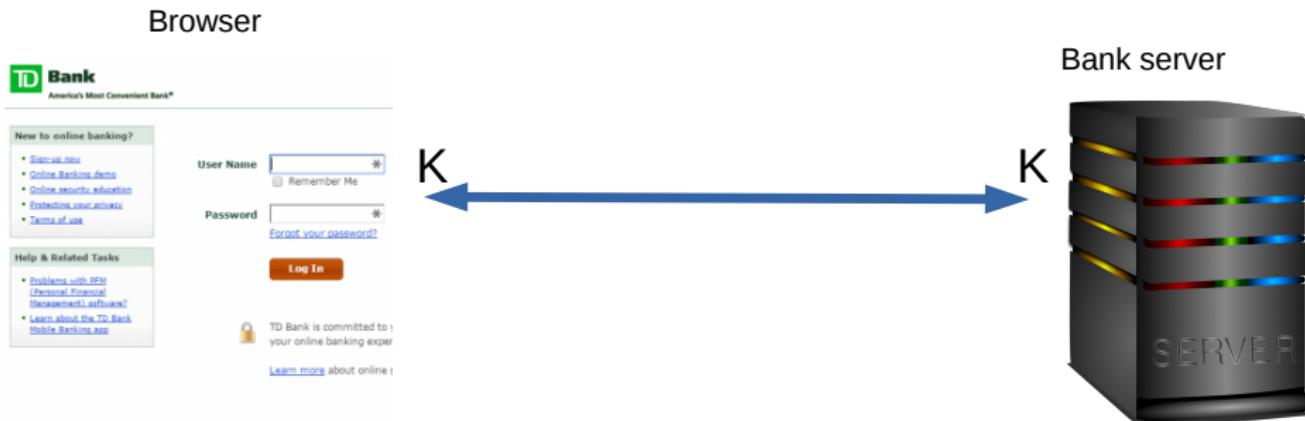
`checksig( vk(x), m, sign(sk(x), r, m) ) = true`

## 2. Authenticating websites and encrypting web traffic

# Encrypting browser-server communication: Encrypt with a shared key K



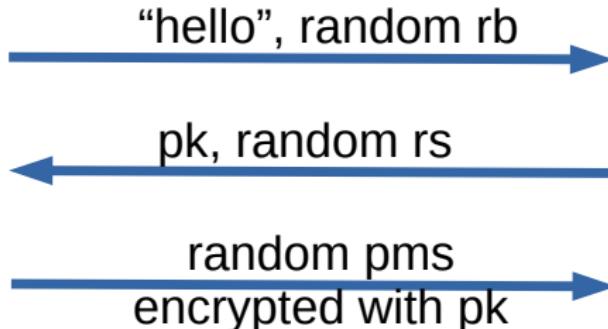
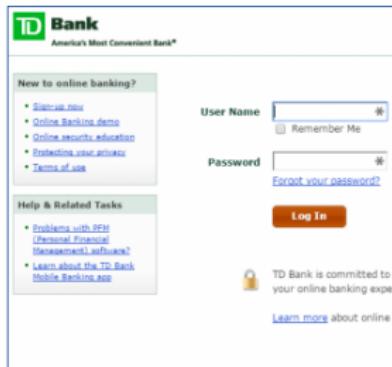
# Encrypting browser-server communication



How to set up the shared key?

# Let's say the server has a public key pk

Browser



Bank server



$$K = f(pms, rb, rs)$$

$$K = f(pms, rb, rs)$$

That's TLS (in abstract form).

# TLS

- In TLS, the session encryption key is established from the server's public key.
- Thus, the browser needs to obtain the server's public key.
- How to achieve that?
  - The server sends it!
  - But how can the browser *authenticate* it?

How can a browser verify that  
a given public key belongs to a given website?

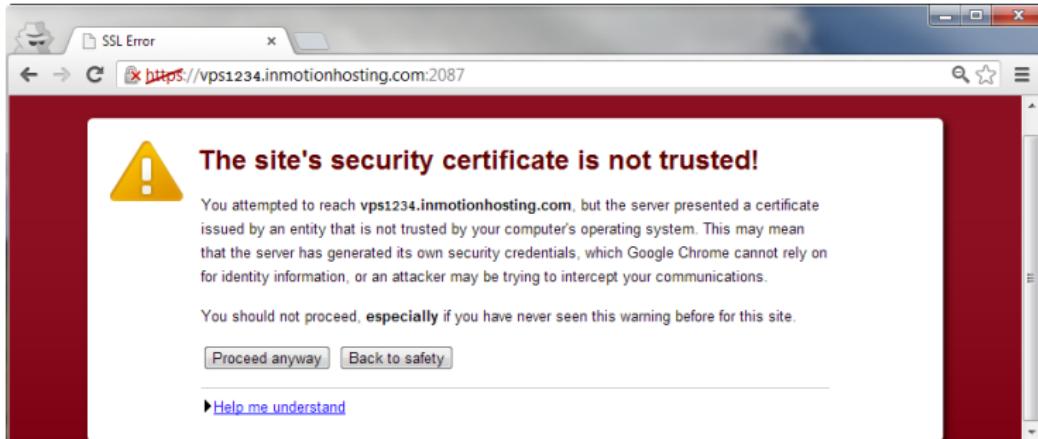
### 3. The Certificate Authority model

# Certificate authorities

- A certificate authority (CA) is a trusted party that asserts that a given public key belongs to a given website.
- The CA signs a certificate  
 $\text{Sign}(\text{sk}_{\text{CA}}, (\text{"HSBC Bank"}, \text{pk}))$
- The browser knows the verification key  $\text{vk}_{\text{CA}}$  needed to verify this signature.
  - It has lots of CA verification keys built in.

# Problems with the CA model

- Usability:  
Incomprehensible certificate warnings



- Insecurity:  
Fake certificates... Any CA can sign an key

# CA model broken

- **Comodo** reseller account broken into and used to fraudulently issue nine certificates, for Google, Microsoft, Yahoo, Mozilla, Skype. (March 2011)
- Wildcard certificate for Google issued by **DigiNotar**'s systems under the control of an attacker. (August 2011)
- MitM attack against Facebook in Syria, by click-through of self-signed certificates, allegedly inserted by Syrian Telecom Ministry. (May 2011)
- **Trustwave** issued a MitM certificate for a company, allowing it to issue valid certificates for any server; used for *data loss prevention* and monitoring of staff use of Gmail and hotmail. (Feb 2012)

# How to avoid attacks like these?

How to ensure that a browser obtains the correct public key for a website?

## Interlude: recalling the twelve principles

1. Secure the weakest link
2. Defend in depth
3. Fail secure
4. Grant least privilege
5. Economise mechanism
6. Authenticate requests
7. Control access
8. Assume secrets not safe
9. Make security usable
10. Promote privacy
11. Audit and monitor
12. Proportionality principle

# Principles that seem relevant to the CA model

## 1. Secure the weakest link

*Significant* weak link: any CA can sign a cert. for any domain

## 2. Defend in depth

*No layering* of security

## 3. Fail secure

CA model *fails insecure*: invalid certs allow click-through

## 4. Grant least privilege

*Not present.* CAs have too much privilege

## 8. Assume secrets not safe.

CA signing keys may not be safe

## 9. Make security usable

Users don't understand certificate warnings

## 11. Audit and monitor

*No opportunity* for domain owners to audit what certificates are being used

## 4. Certificate transparency

# Certificate transparency [Laurie, Kasper, Langley 2012]

**Aim:** ensure that whenever a CA signs a certificate, there is persistent **evidence** of this fact. A CA cannot sign certificates inadvertently/sneakily

**Mechanism:** An *append-only public log* is maintained, consisting of certificates issued by all CAs.

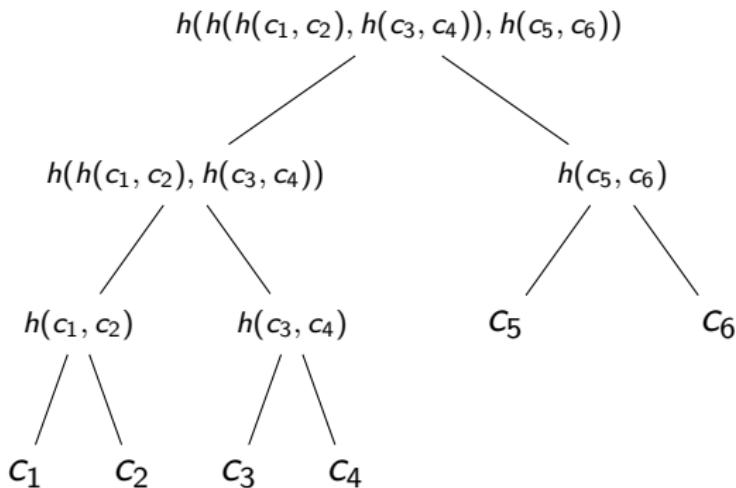
- A certificate is accepted only if it is included in the log  
(The certificate comes with proof that it is included in the log)
- Domain owners can check that certificates about them in the log are really theirs

Users' client software checks the proofs, and also checks that the log is **append-only** and **linear**

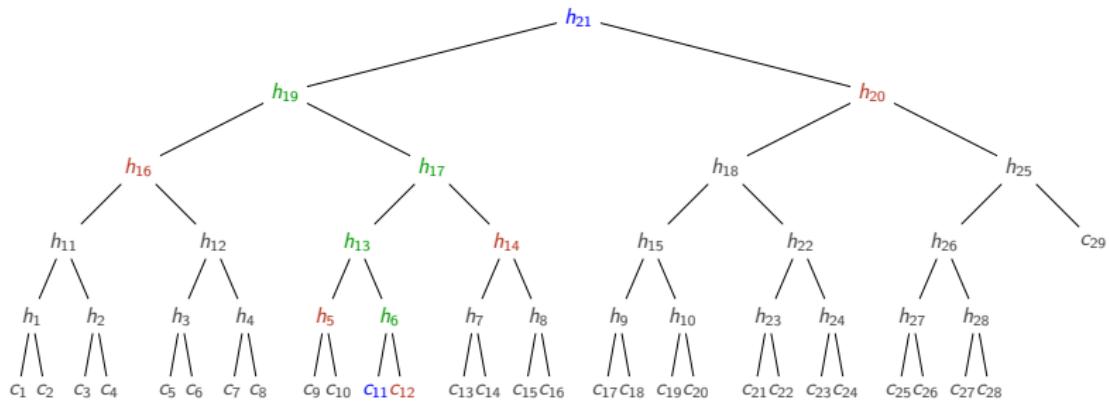
**Status:** IETF draft; RFC. January 2015: Google Chrome checks Google's CT logs for EV certificates.

# Certificate transparency: append-only public log

A log storing certificates  $c_1, c_2, c_3, c_4, c_5, c_6$ :



$\text{prove\_presence}(c_{11}, h_{21})$   
 $= (c_{12}, h_5, h_{14}, h_{16}, h_{20})$



# Algorithms on chronological trees (ChronTrees)

Algorithm	Complexity	Typical size $10^9$ certif.
request_h()	$O(1)$	0.25 KB
prove_presence( $h, cert$ )	$O(\log n)$	2 KB

# Certificate transparency: summary of operation

- A CT log maintains a list of all issued certificates.
- If a certificate is present in the log, it adds assurance as to the validity of the certificate, because . . .
- A domain owner can discover all the certificates about it that are present in the log.
- Browsers consult the log to verify that certificates presented to them are present in the log:
  - The log is organised as a Merkle tree, to facilitate *proofs of presence*.
- Browsers can also verify that the log is being maintained 'append only':
  - The Merkle tree organisation also facilitates *proofs of extension*.

# CT is vulnerable to a *fork attack*



Browser sees a version of the log that contains an **invalid** certificate for the server.



Server sees a version of the log that contains only **valid** certificates for the server.

Browser and Server are seeing different views of the log. This kind of attack could be perpetrated by a powerful adversary, such as a government.

# CT is vulnerable to a *fork attack*



Browser sees a version of the log that contains an **invalid** certificate for the server.



Server sees a version of the log that contains only **valid** certificates for the server.

Browser and Server are seeing different views of the log. This kind of attack could be perpetrated by a powerful adversary, such as a government.

Possible solution: *gossip protocols*

# Revisiting the principles w.r.t. the CT model

## 1. Secure the weakest link

Significant weak link: *any* CA can sign a cert. for *any* domain. *Still true, but the domain owner now knows*

## 2. Defend in depth

*CA and log maintainer provide layering*

## 3. Fail secure

*CT aim is to force hard-fail*

## 4. Grant least privilege

*CA privilege much reduced*

## 8. Assume secrets not safe

*Usefulness to attacker of CA signing key much reduced*

## 9. Make security usable

*Aim for users not to have to see cert warnings - hard fail instead*

## 11. Audit and monitor

*Certificate logging increases transparency*

## 5. Certificate transparency: revocation

# Key revocation

- If the secret key corresponding to a public key is accidentally divulged, or access to it is lost, then the key has to be *revoked*.
  - Revocation can be caused by lost/forgotten passwords, hacked accounts, lost computers or storage, etc.
- To support this, we need more than a *proof of presence*, i.e. a proof that the certificate was once issued.
  - We need *proof of currency* in the log: a proof that the certificate was issued, and has not subsequently been revoked.
  - Another way to see this is: we need *proof of absence* of a revocation in the log.  
 $\text{current} = \text{present} \wedge \neg\text{revoked}$
- Technical challenge: extend CT to support efficient proofs of absence

# Conclusions

- Authenticating websites is a surprisingly hard problem.
- The CA model that we have been using for 20+ years is no longer adequate
- Certificate transparency seems to be the most promising upcoming solution
- Related to the blockchain - also trendy :-)
- Certificate revocation can also be solved in this framework
- Still need to detail a solution for the fork attack