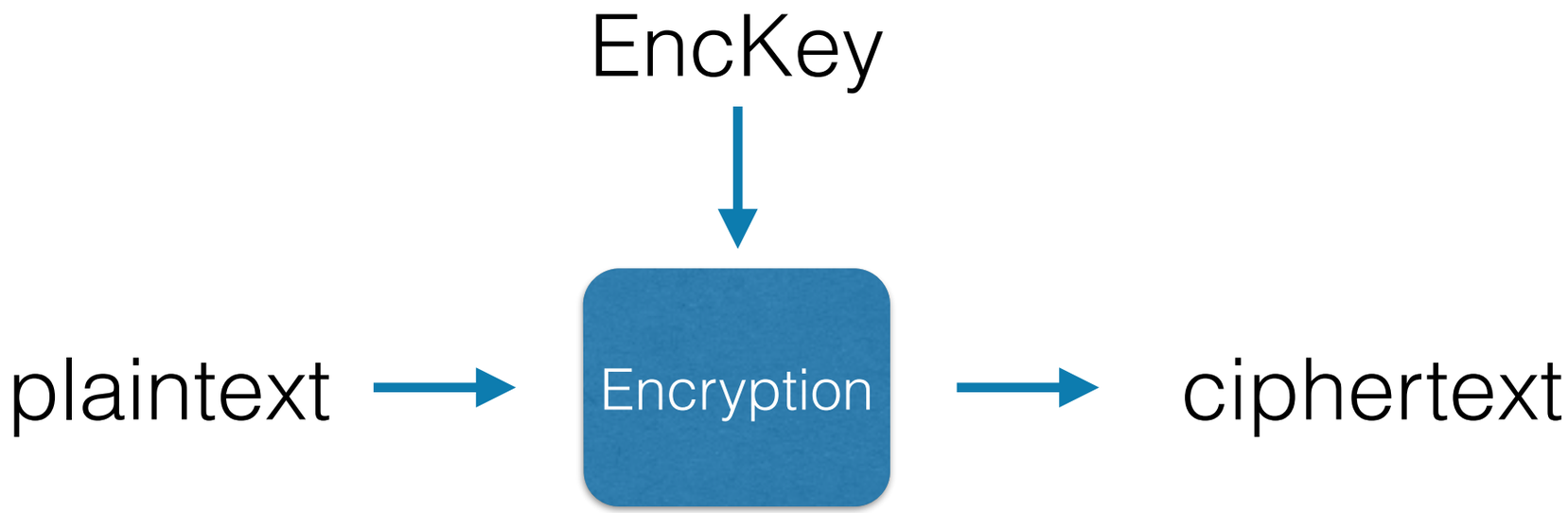
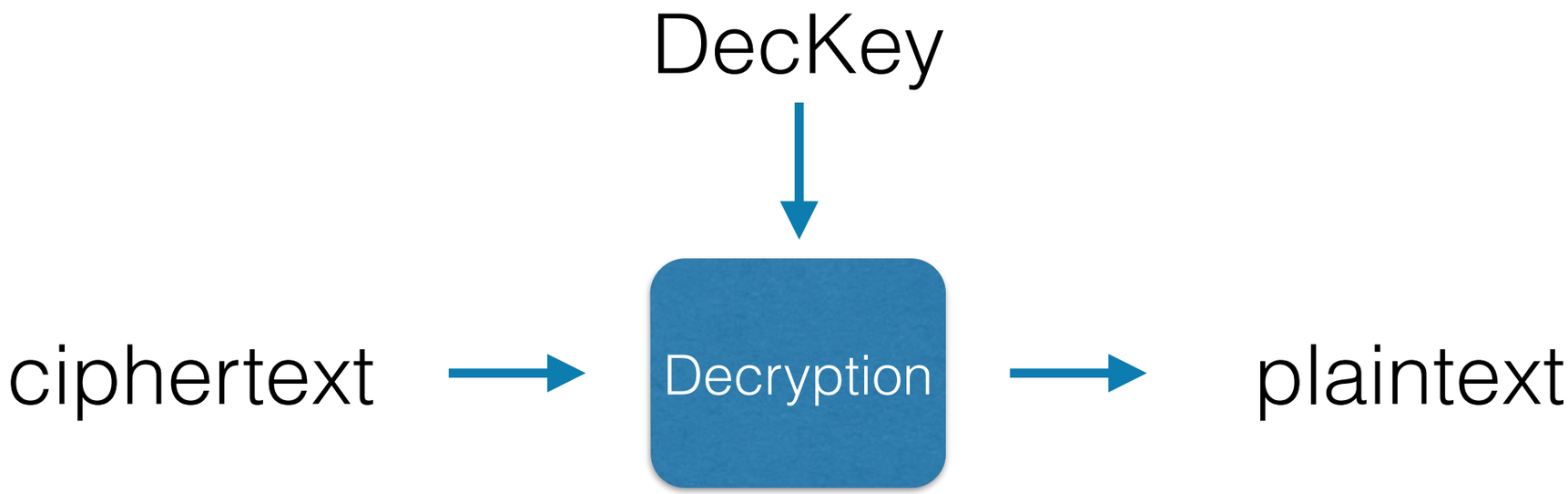
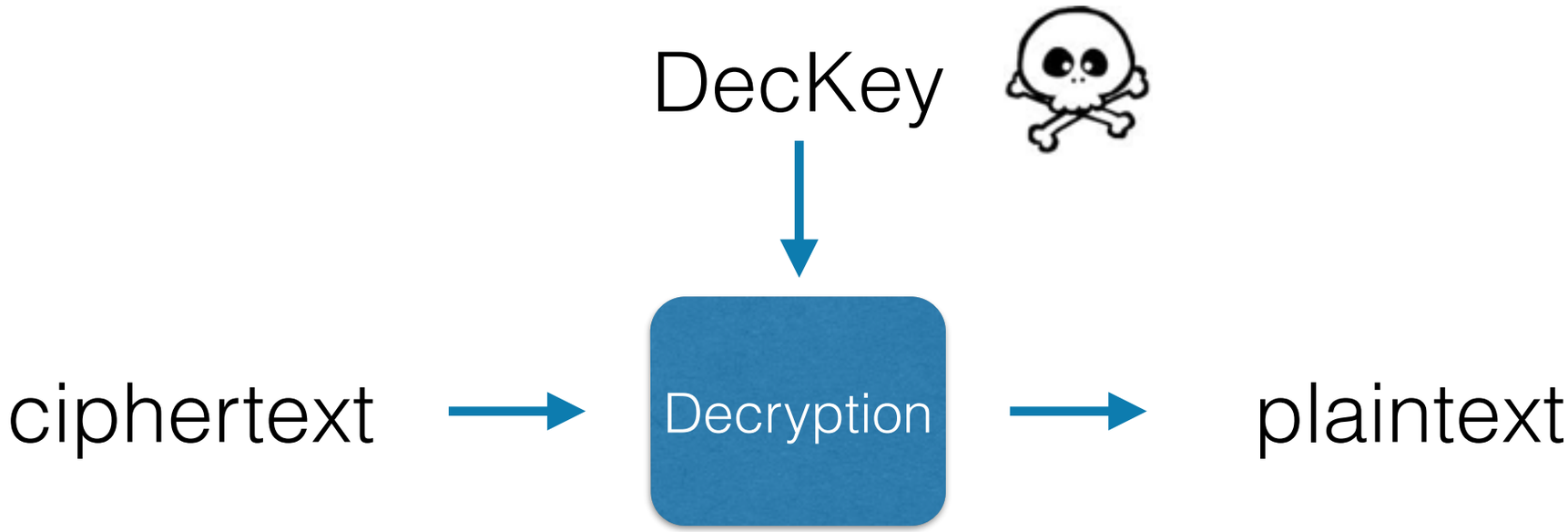
Key Usage Detection

Jiangshan Yu and Mark Ryan
Security seminar
University of Birmingham
Oct.2014


EncKey
plaintext
Encryption
ciphertext

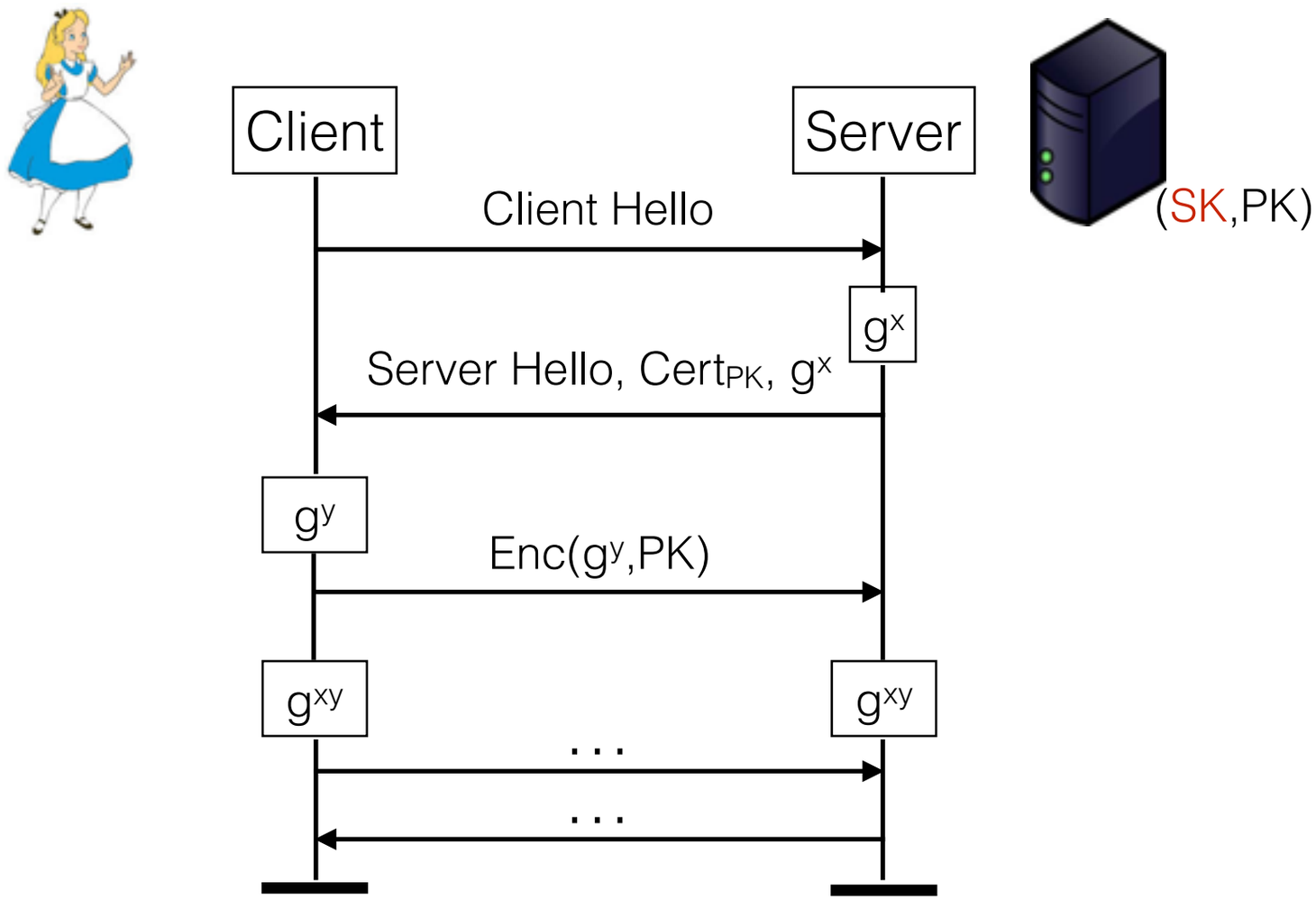


DecKey = EncKey
or
DecKey != EncKey



DecKey = EncKey
or
DecKey != EncKey


SECRETS



Example: Simplified TLS

Let’s assume all crypto primitives are PERFECTLY secure

PKI is also secure

(e.g. Perspective, convergence, DANE, CT, AKI, ARPKI, DTKI, …)

Basically … Everything is assumed to be secure

But …What if the SK is compromised

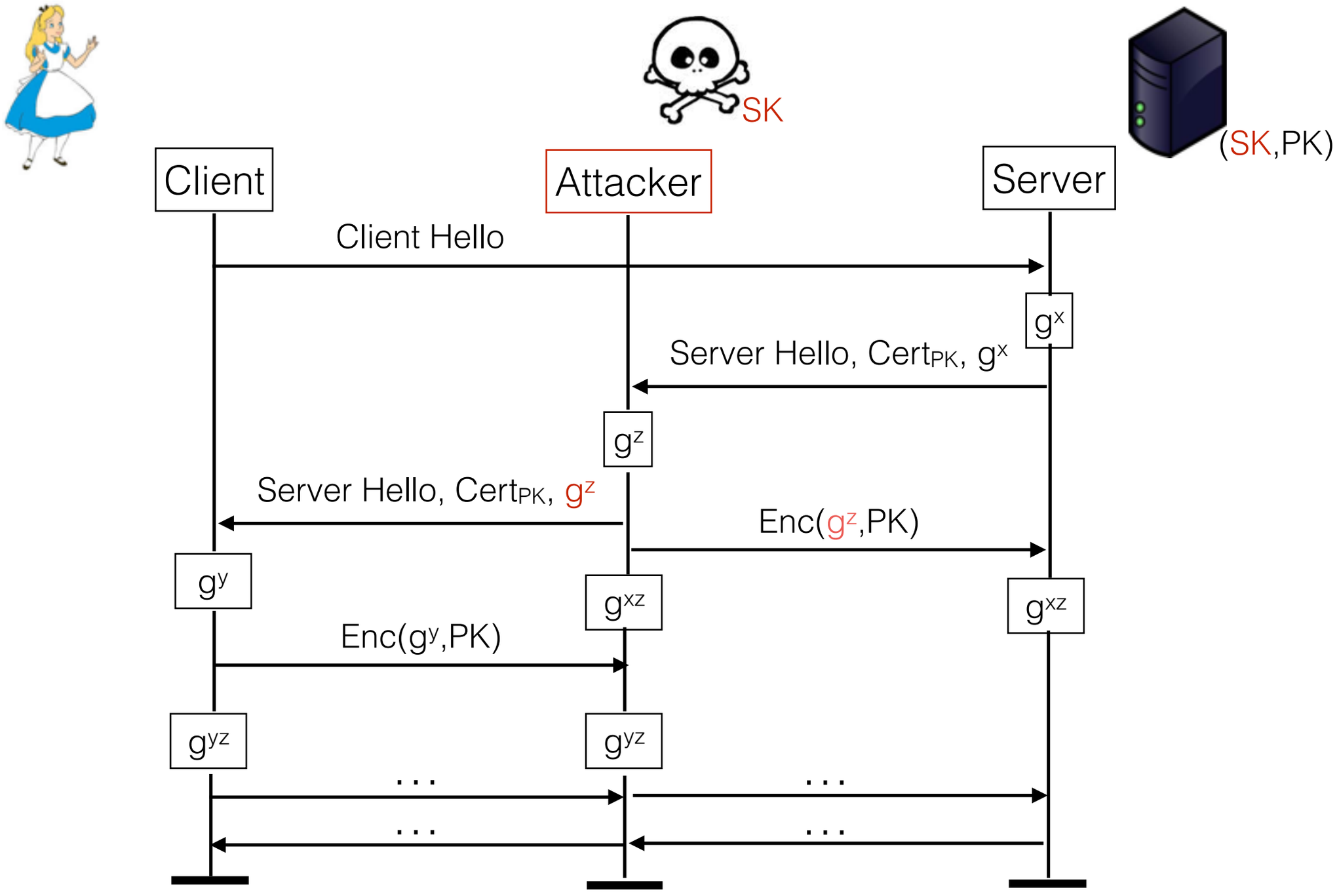
SK


SK


SK

SK

SK



Example: a MITM attack on the simplified TLS

Can web servers take care of their keys?

Schneier on Security

Blog | Newsletter | Books | Essays | News | Schedule | Crypto | About Me

← FBI Asks for Cryptanalysis Help — 34 SCADA Vulnerabilities Published →

Comodo Group Issues Bogus SSL Certificates

This isn't good:

> The hacker, whose March 15 attack was traced to an IP address in Iran, compromised a partner account at the respected certificate authority Comodo Group, which he used to request eight SSL certificates for six domains: mail.google.com, www.google.com, login.yahoo.com, login.skype.com, addons.mozilla.org and login.live.com.

https://www.schneier.com/blog/archives/2011/03/comodo_group_is.html

Compromised CA Certificate Attacks

The Attack

If a Certificate Authority (CA) is compromised, all trust associated with the certificates issued by that CA are compromised, too. There are a number of ways the Certificate Authority can be compromised. One way is for the attacker to steal the private key of a root CA, an intermediate CA, or an issuing CA. If this happens the attacker can sign certificates as if he was the entity whose private key was stolen. Another way the Certificate Authority can be compromised is by the attacker gaining access to issue arbitrary certificates from the issuing CA. This was the case with the infamous DigiNotar compromise.

Two other compromises include: impersonation and forged digital signatures. These compromises are enabled by the extensive trust associated with public CA's. In these compromises, an attacker is issued a certificate by a subordinate public CA of a trusted root CA (such as VeriSign). Since most servers and browsers trust CA's such as VeriSign, the attacker's certificate will appear valid.

The Tools

Tools for the attack include: SSLsniff

https://www.vidder.com/resources/attacks/compromised-ca-certificate.html

The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

http://heartbleed.com

The 10 most common Windows security vulnerabilities

http://searchenterprisedesktop.techtarget.com/tip/The-10-most-common-Windows-security-vulnerabilities

http://news.sky.com/story/1341877/apple-mac-bug-is-bigger-than-heartbleed

Topic: Cloud

Follow via:

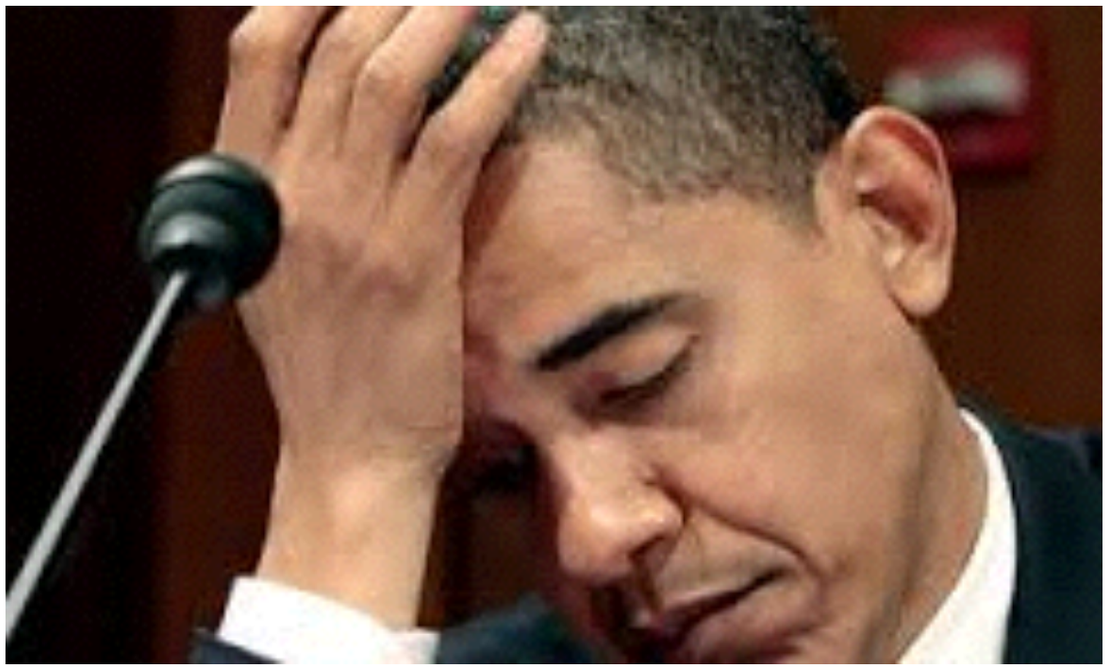
Shellshock makes Heartbleed look insignificant

Summary: *The new vulnerability in the Bash shell is the worst we've seen in many years. No software on critical systems can be assumed as safe.*

http://www.zdnet.com/shellshock-makes-heartbleed-look-insignificant-7000034143/

:(

Key Revocation



A Simple Proposal — Key Usage Detection

Attacker Model:

we consider an adversary who can get all (long-term and short-term) secrets of victims, but only for a limited period. The attacker may get control of the victim's device repeatedly, but not continuously.

Attacker Model:

we consider an adversary who can get all (long-term and short-term) secrets of victims, but only for a limited period. The attacker may get control of the victim's device repeatedly, but not continuously.

Main security properties:

- forward secrecy

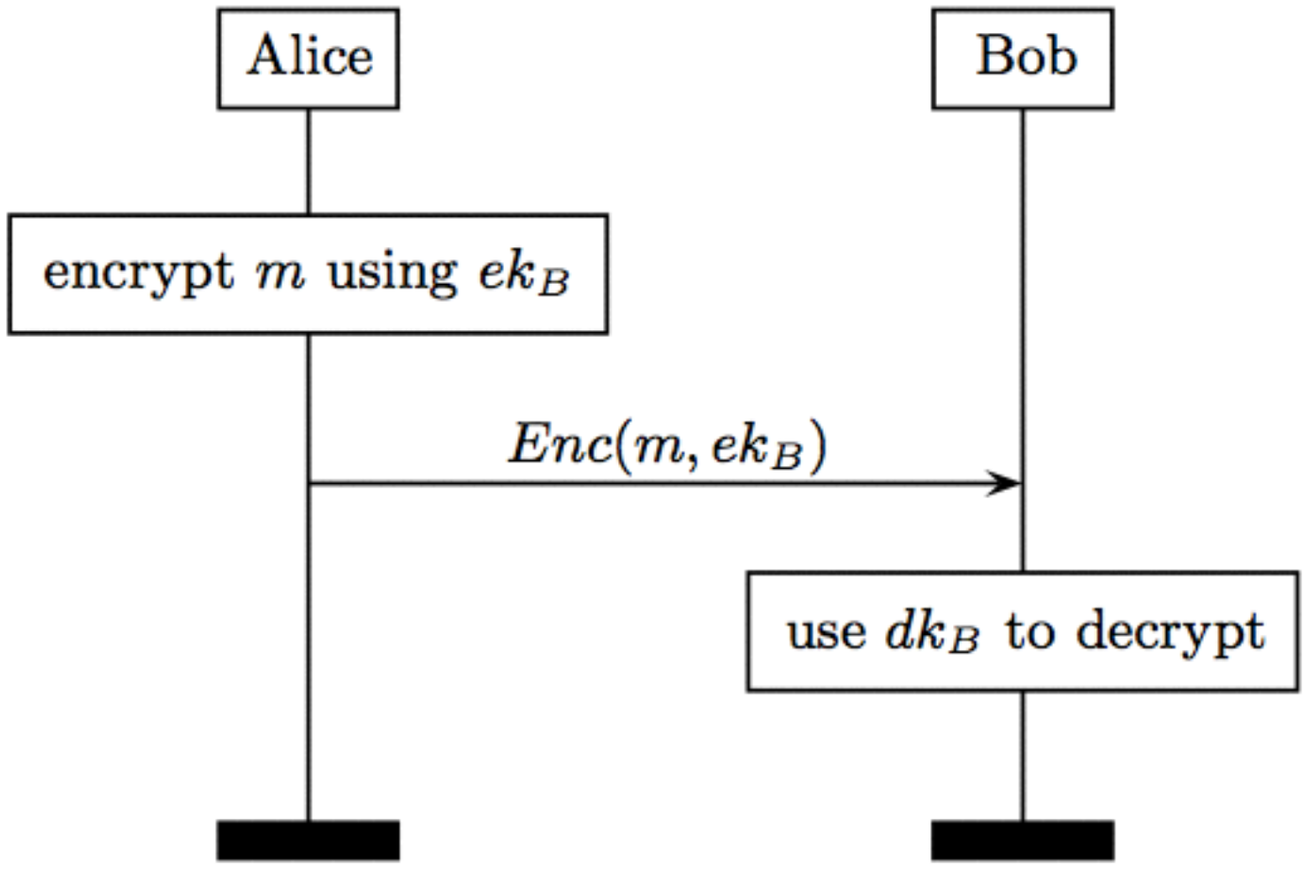
Main security properties:

- forward secrecy
- auditable-backward secrecy

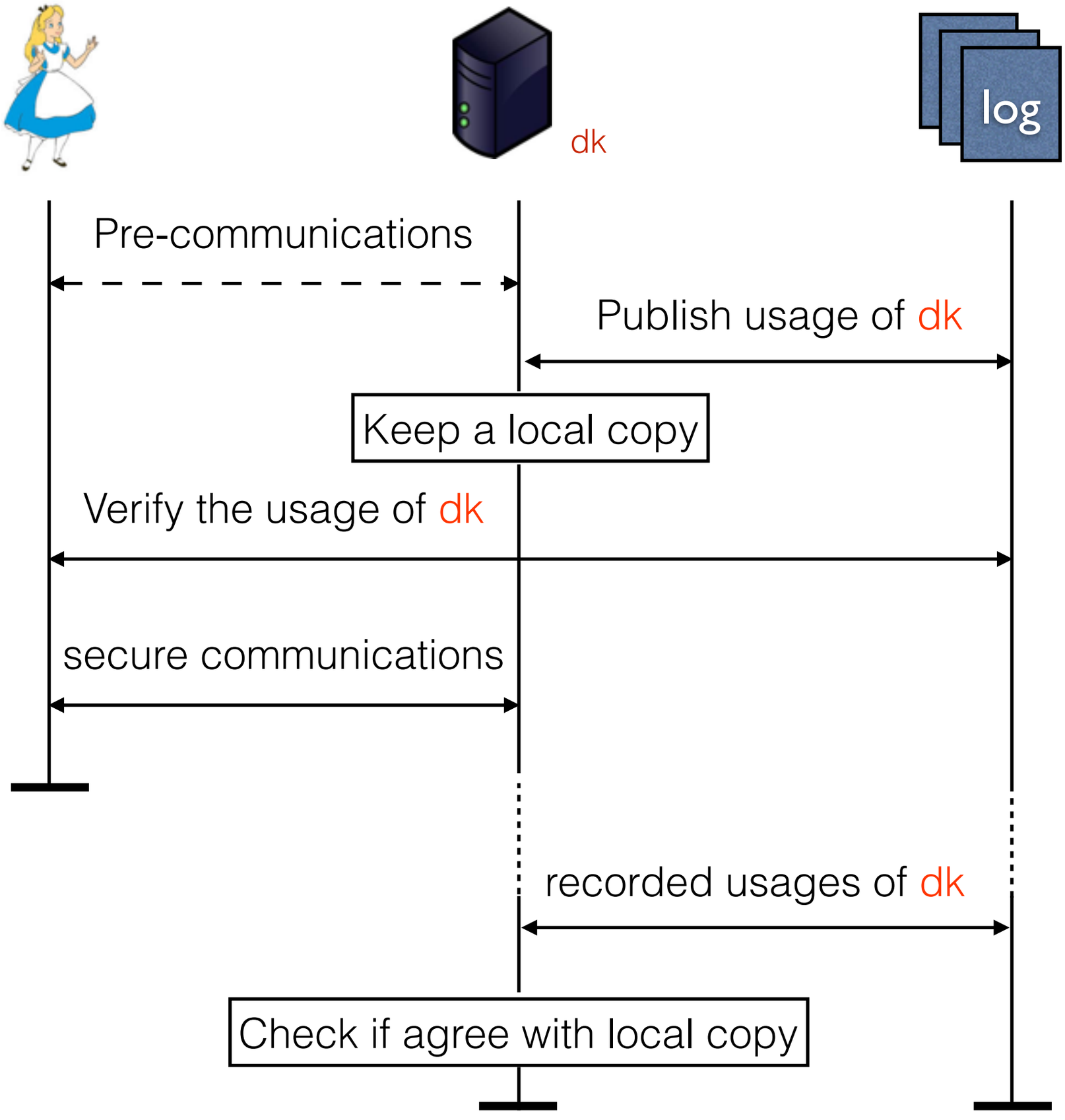
Main security properties:

- forward secrecy
- auditable-backward secrecy

- broken
- secure



The current protocol using a/symmetric key encryption


dk
log
Pre-communications
Publish usage of dk
Keep a local copy
Verify the usage of dk
secure communications
recorded usages of dk
Check if agree with local copy

Main difficulty:

How to commit the usage of dk in the log?

How to commit the usage of dk in the log?

How to commit the usage of dk in the log?

- generate a contribution of session key establishment (e.g. g^x in DH)

How to commit the usage of dk in the log?

- generate a contribution of session key establishment (e.g. g^x in DH)
- issue a commitment on the contribution by using dk

How to commit the usage of dk in the log?

- generate a contribution of session key establishment (e.g. g^x in DH)
- issue a commitment on the contribution by using dk
- publish the commitment in the log

How to commit the usage of dk in the log?

- generate a contribution of session key establishment (e.g. g^x in DH)
- issue a commitment on the contribution by using dk
- publish the commitment in the log
- use the established session key to for message encryption/decryption

How to generate commitments using symmetric key dk?

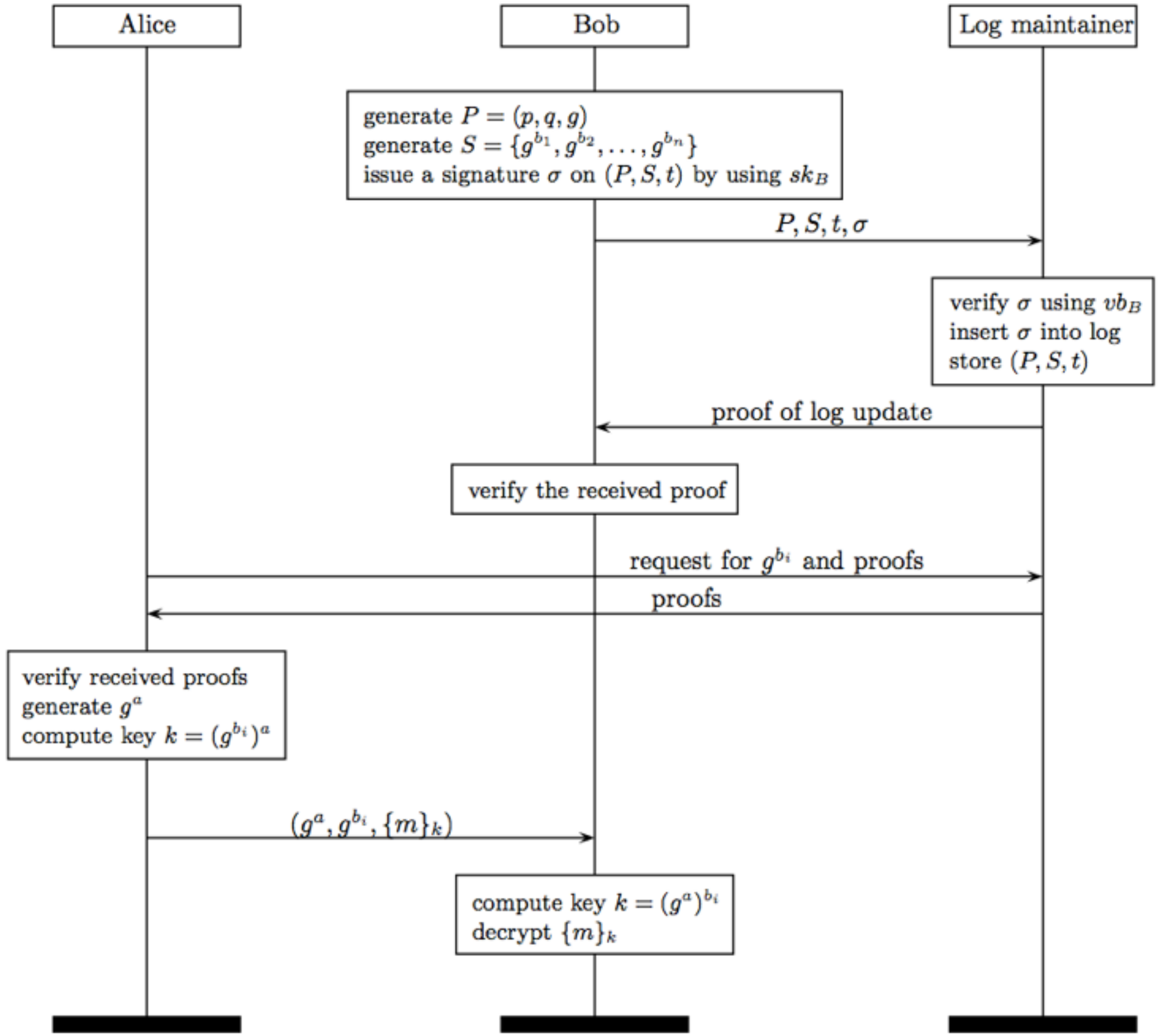
How to generate commitments using symmetric key dk?

HMAC, AES, etc.

How to generate commitments using asymmetric key dk?

How to generate commitments using asymmetric key dk?

Signature, proof of knowledge, etc.



overview of our proposal

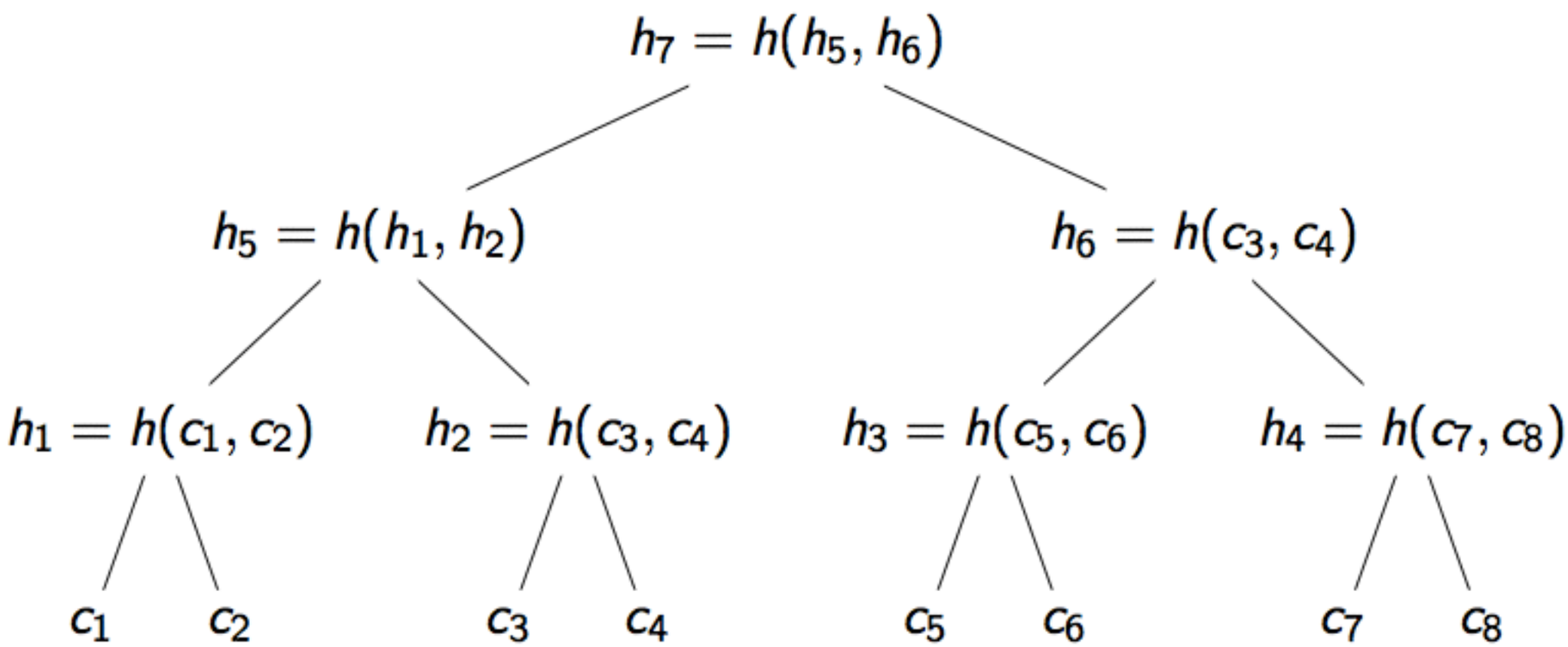
Log structure

Log structure

1. Merkle, R.C.: A digital signature based on a conventional encryption function. CRYPTO. (1987)
2. Laurie, B., Langley, A., Kasper, E.: Certificate Transparency. RFC 6962. (2013)
3. Kim, T.H.J., Huang, L.S., Perrig, A., Jackson, C., Gligor, V.: Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure. WWW. (2013)
4. Ryan, M.D.: Enhanced certificate transparency and end-to-end encrypted mail. in network and distributed system security. NDSS. (2014)
5. Yu, J., Cheval, V., Ryan, M.: DTKI: a new formalized PKI with no trusted parties. IACR Cryptology ePrint Archive. (2014)

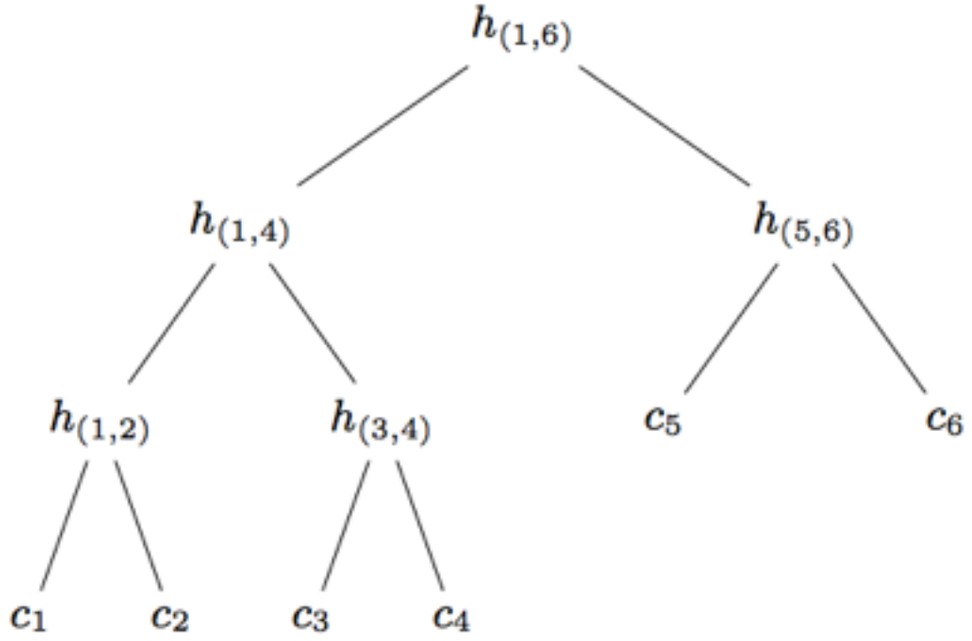
The log structure supports:

- Proof of presence
- Proof of extension

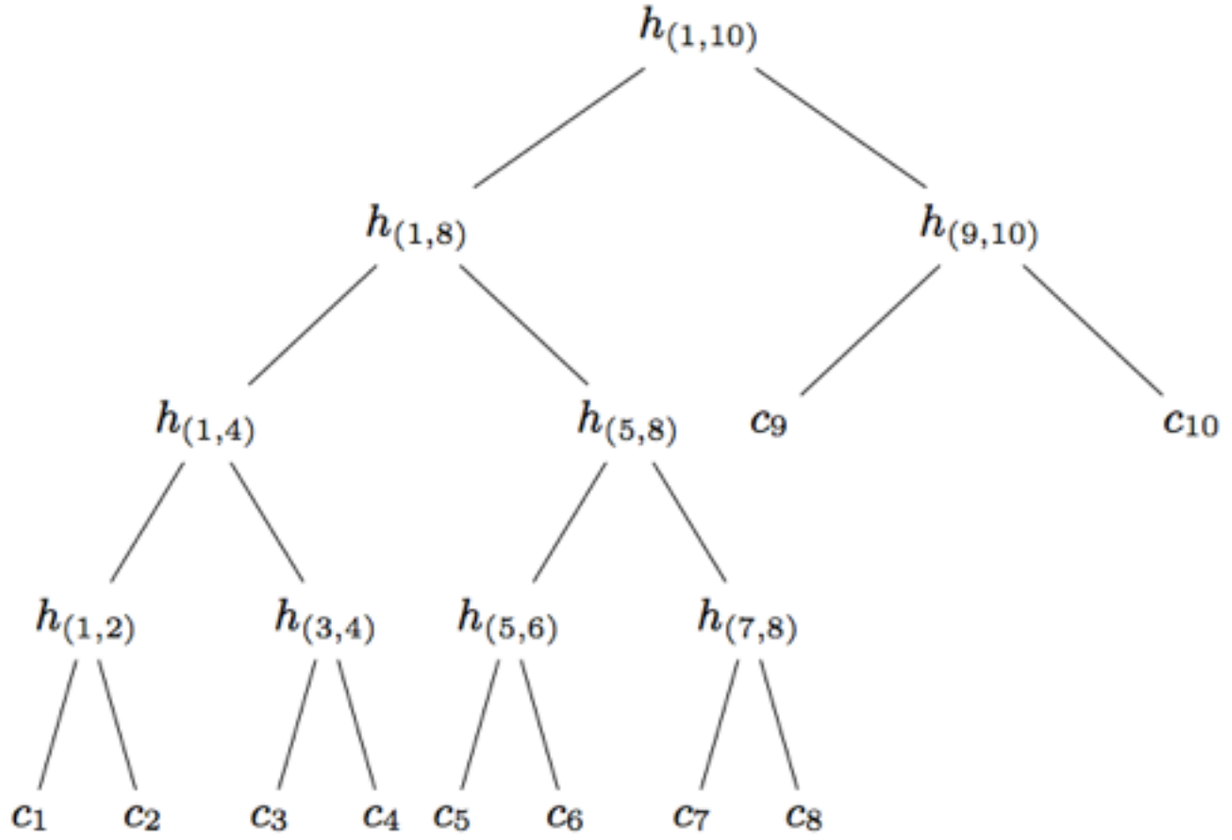


Merkle tree, in which data are stored in chronological order

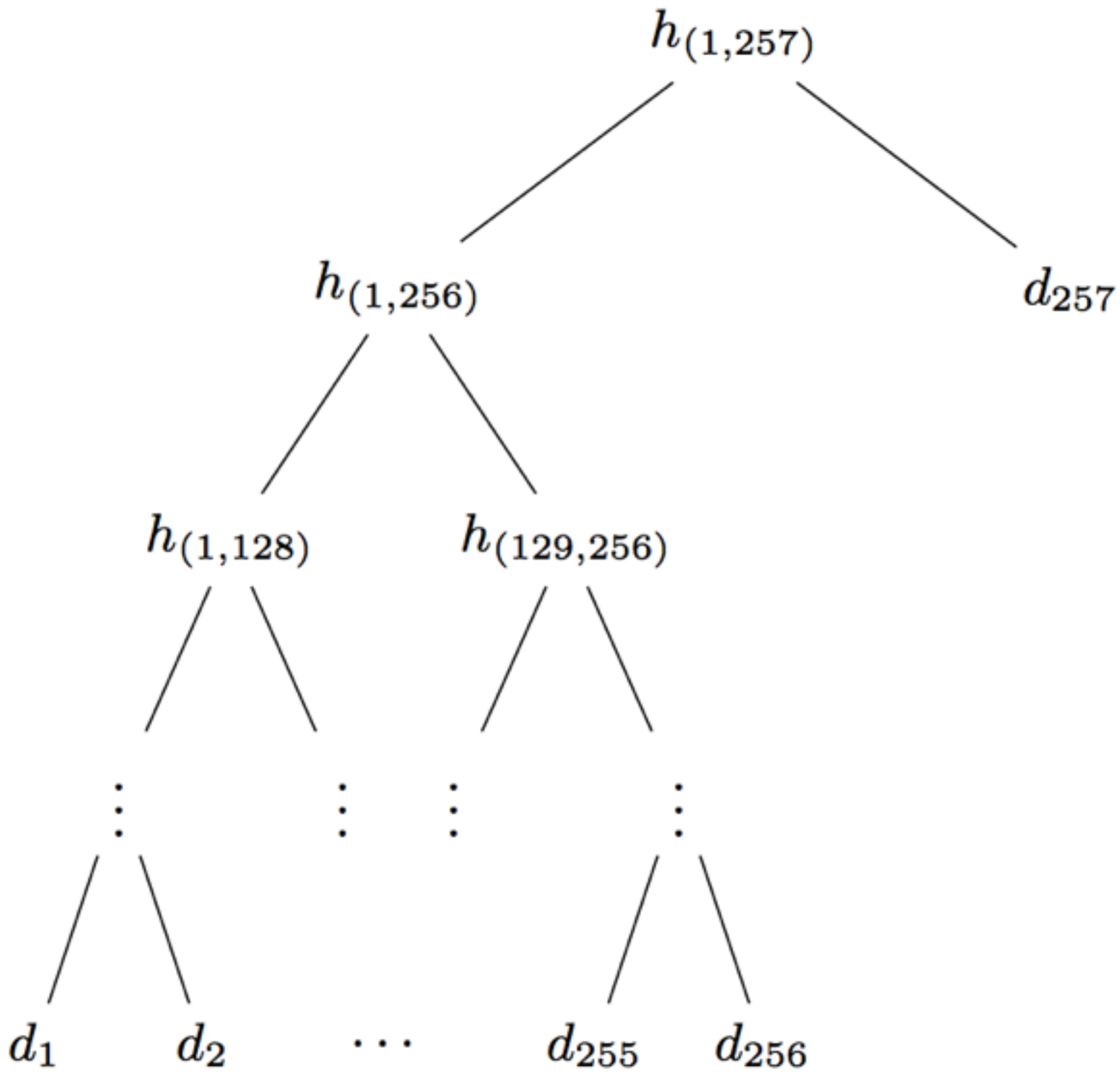
(ChronTree)



(a) Merkle Tree T_a



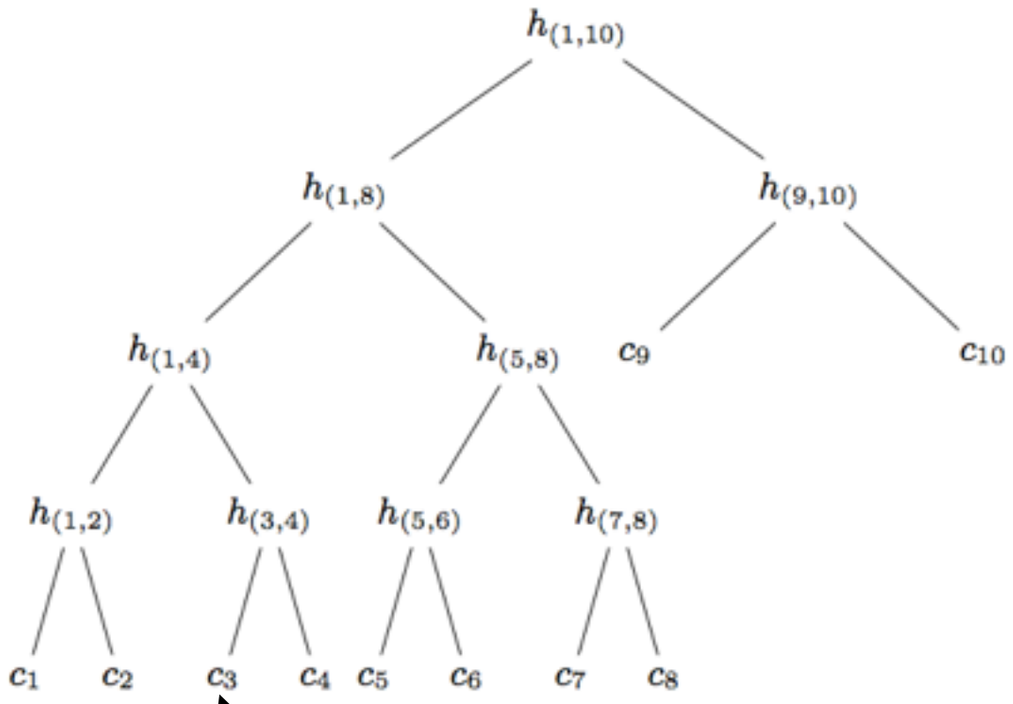
(b) Merkle Tree T_a



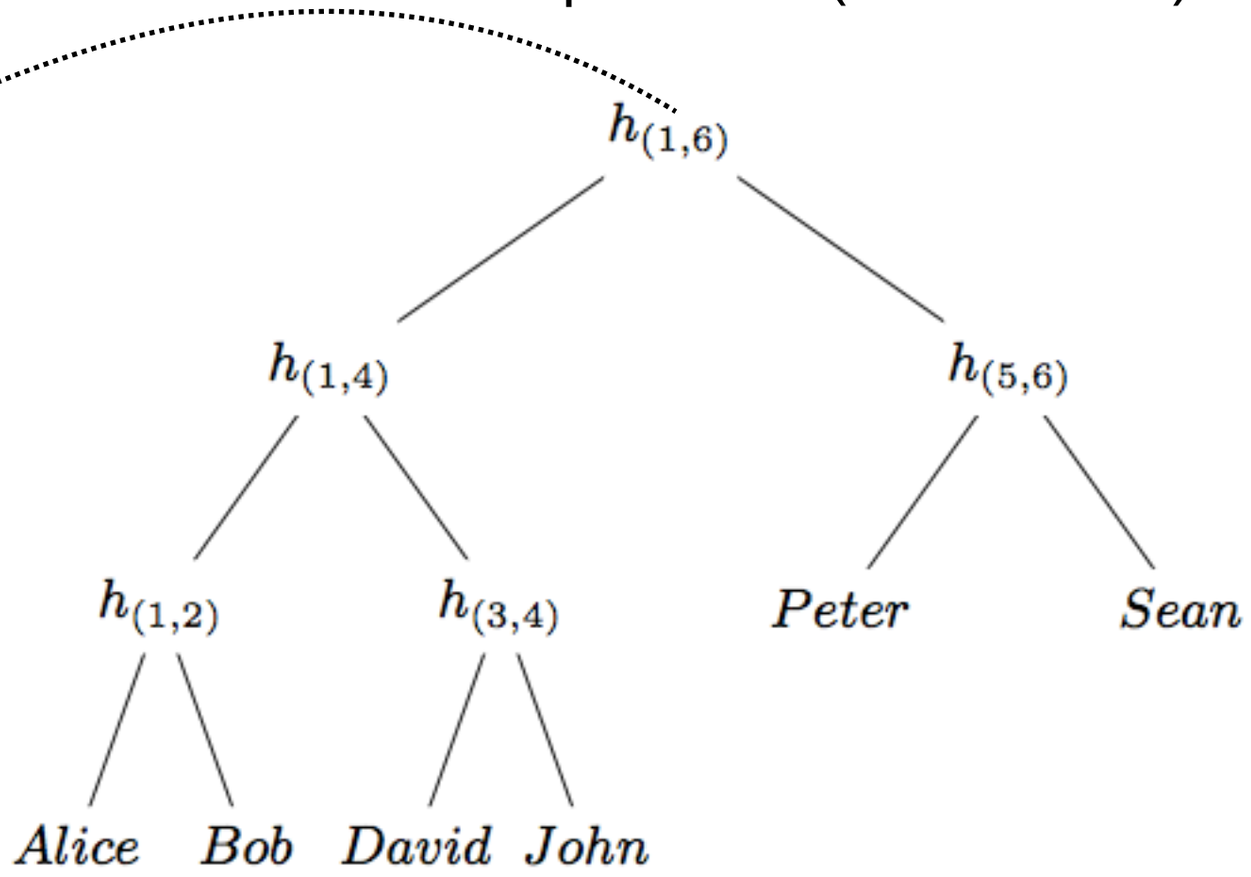
Merkle tree, in which data are stored in lexicographic order

(LexTree)

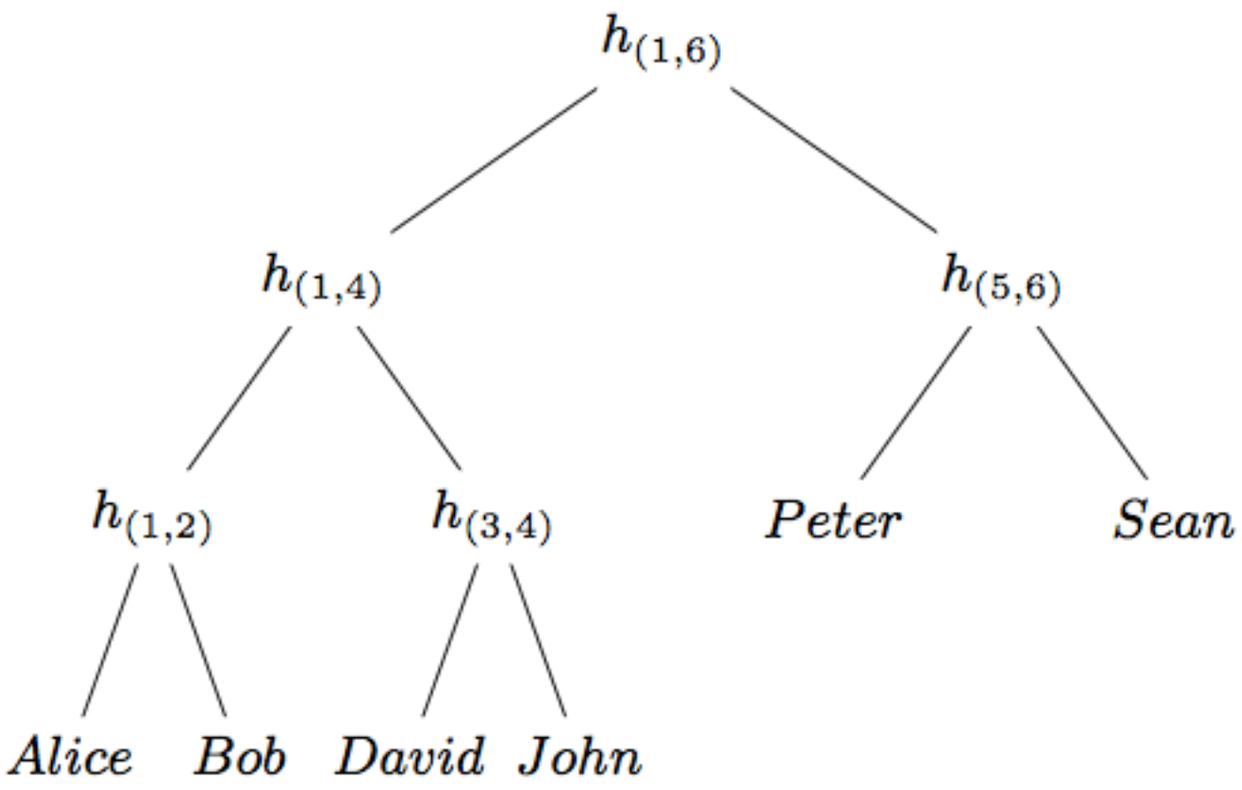
Log (ChronTree)



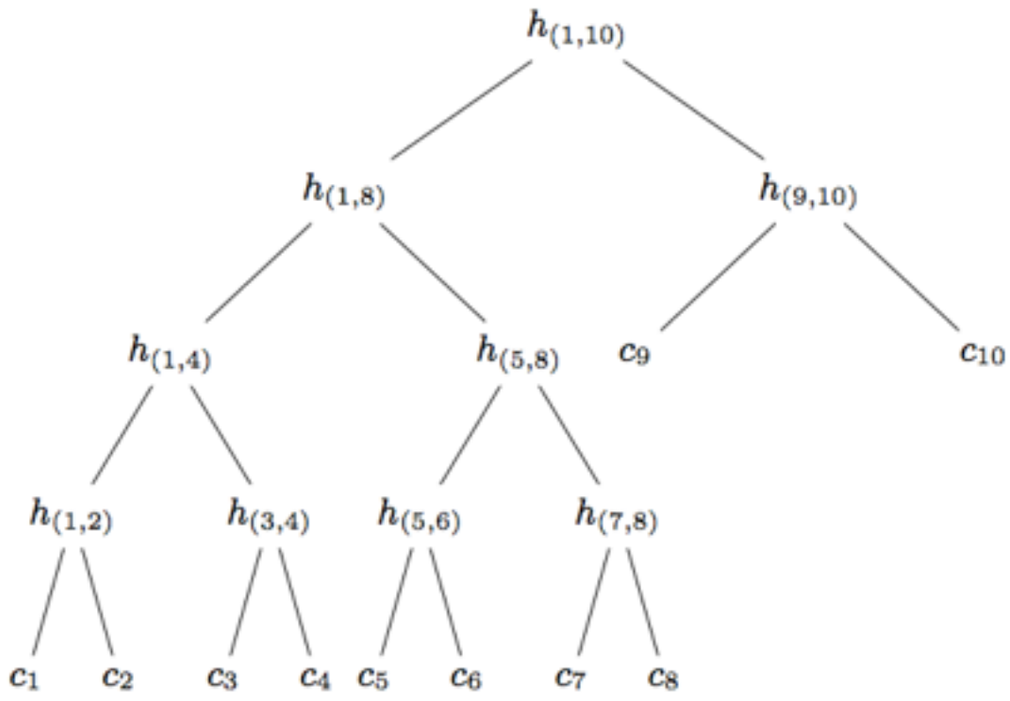
The 3rd update (LexTree)



The 3rd update (LexTree)



ChronTree of Bob



Security

Let δ be the lifetime of short-term secrets.

Suppose Bob's system is compromised at time t_i and patched at time t'_i. Then, the attacker cannot recover the plain-text of the exchanged messages without being readily detected at anytime other than in the time period $(t_i - \delta, t'_i + \delta)$.

- broken
- secure

Thank you