Lecture Notes

Esecurity: secure internet & evoting

Michael Nüsken

b-it

(Bonn-Aachen International Center
for Information Technology)

Summer 2010

esec
12.04.10
1

Email

Goal: (originating before 1982,
ie. long before the Internet)

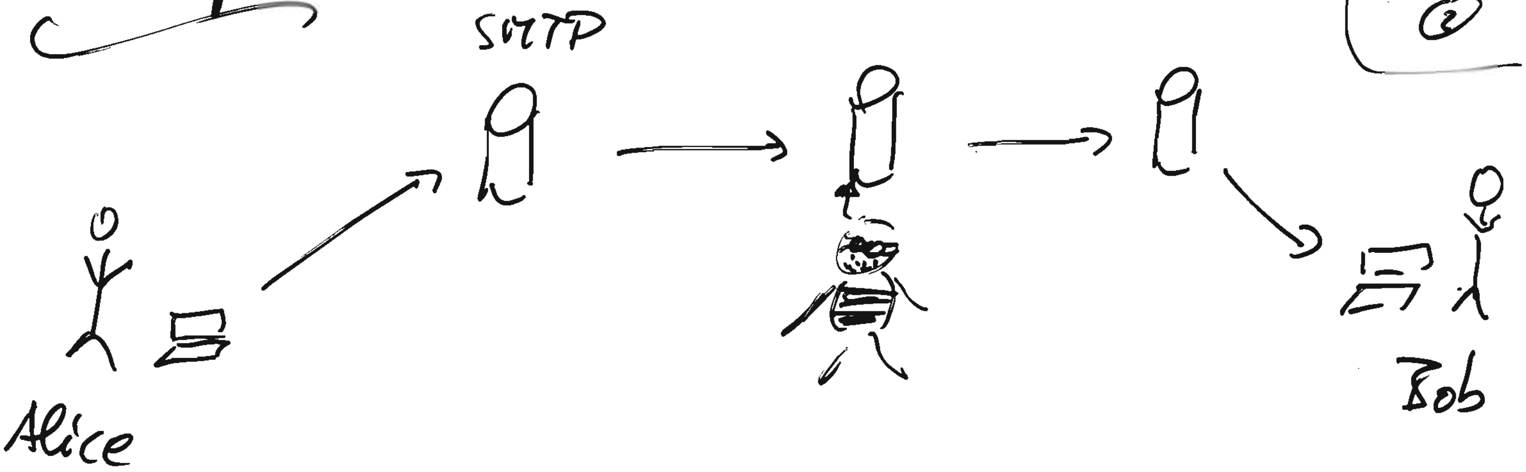
- transfer messages of varying size
- text messages
  not pictures, or even films
- easy, fast
- connects geographically distributed parties
- independent of sender & recipient location.

Format:

- pure text, electronic
- simple format:
  <header> ⟶ <keyword>: <info>
  <blank line>
  <body>

Thunderbird: Ctrl+U
shows raw mail text...

Transport



Security? Goals now:

- identify sender (authenticate)
- encryption identify receiver (confidential!)
- prevent changes of content (integrity)
- prevent even ~~know~~ existence of messages to be known (message flow confidentiality)
- signature
- sender cannot deny the content. (non-repudiation)
- Proof of submission
- Proof of delivery
- Anonymity

header

```
Return-Path: <08ws-soti-admin@bit.uni-bonn.de>
X-Original-To: nuesken@math.upb.de
Delivered-To: nuesken@math.upb.de
[...]
Received: by postfix.iai.uni-bonn.de (Postfix, from userid 13020)
        id 94C365C834; Mon,  3 Nov 2008 21:10:04 +0100 (MET)
X-Sieve: cmu-sieve 2.0
X-IAI-Env-From: <08ws-soti-admin@bit.uni-bonn.de> : [131.220.8.1]
Received: from uran.iai.uni-bonn.de (uran.iai.uni-bonn.de [131.220.8.1])
        by postfix.iai.uni-bonn.de (Postfix) with ESMTP
        id 97F4F5C829; Mon,  3 Nov 2008 21:10:03 +0100 (MET)
        (envelope-from 08ws-soti-admin@bit.uni-bonn.de)
        (envelope-to VARIOUS) (2)
        (internal use: ta=0, tu=1, te=0, am=-, au=-)
Delivered-To: 08ws-soti@alias.informatik.uni-bonn.de
X-IAI-Env-From: <first.family@uni-bonn.de> : [80.136.68.129]
Received: from [192.168.178.46] (p50884481.dip.t-dialin.net [80.136.68.129])
        by postfix.iai.uni-bonn.de (Postfix) with ESMTP
        id A1CCC5C829; Mon,  3 Nov 2008 21:09:55 +0100 (MET)
        (envelope-from first.family@uni-bonn.de)
        (envelope-to VARIOUS) (2)
        (internal use: ta=1, tu=1, te=1, am=P, au=first.family)
Message-ID: <490F5A8B.6000205@informatik.uni-bonn.de>
Date: Mon, 03 Nov 2008 21:09:47 +0100
From: First Family <first.family@uni-bonn.de>
Reply-To: first.family@uni-bonn.de
User-Agent: Thunderbird 2.0.0.17 (Windows/20080914)
MIME-Version: 1.0
To: 08ws-soti@bit.uni-bonn.de
Subject: [08ws-soti] 1234567
X-Enigmail-Version: 0.95.7
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
Sender: 08ws-soti-admin@bit.uni-bonn.de
Errors-To: 08ws-soti-admin@bit.uni-bonn.de
X-BeenThere: 08ws-soti@bit.uni-bonn.de
X-Mailman-Version: 2.0.4
Precedence: bulk
[...List-Stuff...]
X-Virus-Scanned: by mailscan-system at math.uni-paderborn.de
X-Spam-Status: No, hits=0.2 tagged_above=-999.0 required=4.0 tests=AWL,
        BAYES_00, DNS_FROM_SECURITYSAGE, SPF_PASS, SUBJ_HAS_UNIQ_ID,
        UNIQUE_WORDS
X-Spam-Level:
```

loss security

DNS → 131.?.?.?

← blank line

body


```
-----BEGIN PGP MESSAGE-----
Charset: UTF-8
Version: GnuPG v1.4.9 (MingW32)
Comment: Using GnuPG with Mozilla - http://enigmail.mozdev.org

hQIOA8SRdzc1IdlqEAf/VqwMFWs1Y2rqD0AQgBjJAyVWshp6TnEFutXOEloM4q4z
CVtNAium3o2+6R3bToYgx7NIetmiQWsRm7o5QWmIeDKu6zu2ogvn275ik71vBAKk
0/M+IfU12WSjpmYDZm62R2iAjwlQy6BbLbPeGXJ/AICm65mgajUT/mum8PA8ako6
EezCwYpbS3A0V0xHopKWDWtc9iUBaIsGR9xLozvcVyXXWMCJSV/BAHewoTFD8U57
vnMU0oSp/j8VjI+kp6koY86MJoNplcUUYG5j+IHnuJpfpIbxs2c5cNwYLKFuvZrV
RpnjoDq/61ATmssidZEw5mF4/utOG913ftKoCdXpGAf9Fzul4wPGUFOzcATLX4Ef
Q+I+x60keFC4K+mIwefsZHdhbT/XtilkeoFCtaHtvwWaqTuaSfxRnlaJshQzwHxL
[...]
aHvqZs9s5+264Q0yUgB8i7AVq6d64JL8lg1h3vKEcDdFFUbSlgEYjsQ0zFI4UK0i
H+xRNHEYaC8UN1EYbul0lx1MZxz3VQ8bneX7cWmuYggkYDM0XUWfX6OP3CKoCWoU
OmZbZWGzH+Il2nzeRO9/TOtHfF5enDO2yuEF3Fr6flFDjlsZIFDq4jdrZy6ucMuO
o2AR6QwuWJQO37KIiJg1ngcfA+SO+Mbdg803wuMH3ORVMNc1ejo5DYRlxw==
=suKP
-----END PGP MESSAGE-----
```


```
_______________________________________________
08ws-SotI mailing list
08ws-SotI@bit.uni-bonn.de
https://mailbox.iai.uni-bonn.de/mailman/listinfo.cgi/08ws-soti
```

Send Mail Transfer Protocol

Example of the SMTP Procedure

This SMTP example shows mail sent by Smith at host Alpha.ARPA, to Jones, Green, and Brown at host Beta.ARPA. Here we assume that host Alpha contacts host Beta directly.

```
S: MAIL FROM:<Smith@Alpha.ARPA>
R: 250 OK

S: RCPT TO:<Jones@Beta.ARPA>
R: 250 OK

S: RCPT TO:<Green@Beta.ARPA>
R: 550 No such user here

S: RCPT TO:<Brown@Beta.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: <CRLF>.<CRLF>
R: 250 OK
```

header + body of the email

The mail has now been accepted for Jones and Brown. Green did not have a mailbox at host Beta.

Example 1

---

esecurity
17.04.10
1

A few technicalities

- everything around around email must stay simple. (easy to use)
- want to receive any mail
- relay (or forward) mail
- address info MUST be included and legible

↳ DNS service supplies information about the topology (¿ Security? ¡)

↳ SMTP = Simple Mail Transfer Protocol specifies details

esecurity
14.04.10
2

SECURITY

content, semantics

intentional attacks

RELIABILITY (safety)

format, syntax

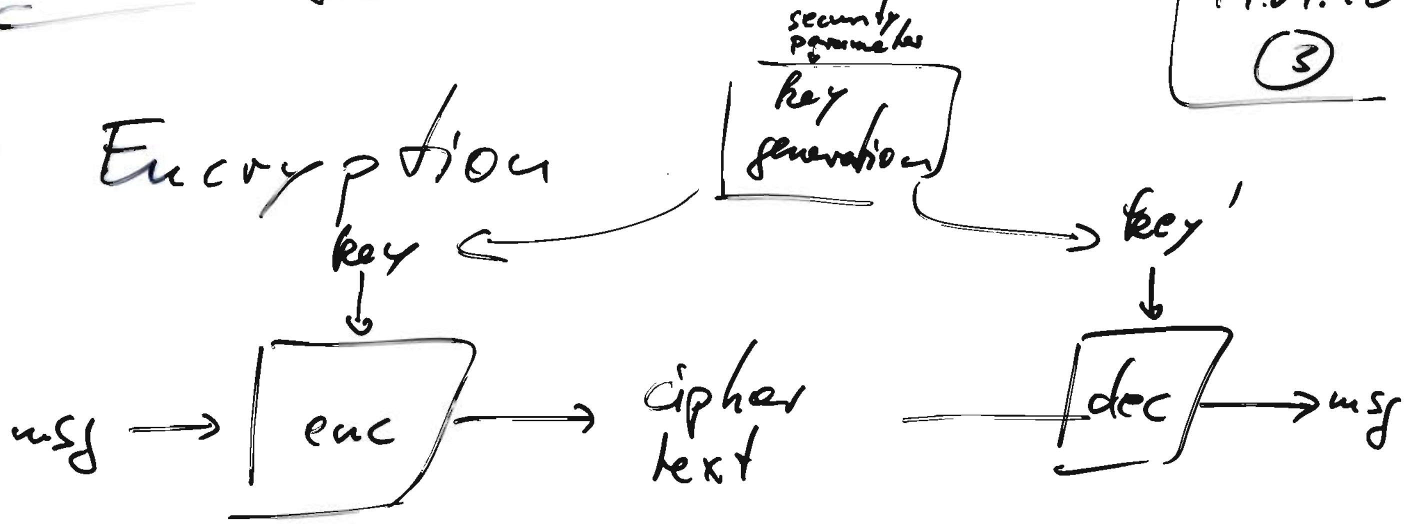
random attacks

DoS

Attack (vs. Email)	Defence	
DoS	–	SAFETY
SPAM	Greylisting Filtering	SAFETY
Phishing	Knowledge	EDUCATION
Read mails	Encryption	(CRYPTO)
Send malware (virus, trojan, worm, ...)	(Firewall) · Antivirus · Knowledge	
Send mails from ~~wrong~~ faked sources Send mails again	Signatures	(CRYPTO)
Changing content	Signature	(CRYPTO)

Technology

(1) Encryption



Symmetric case: key' = key

Public-key case: key and key' are somehow related, but computing key' from key is difficult.

public key

private key

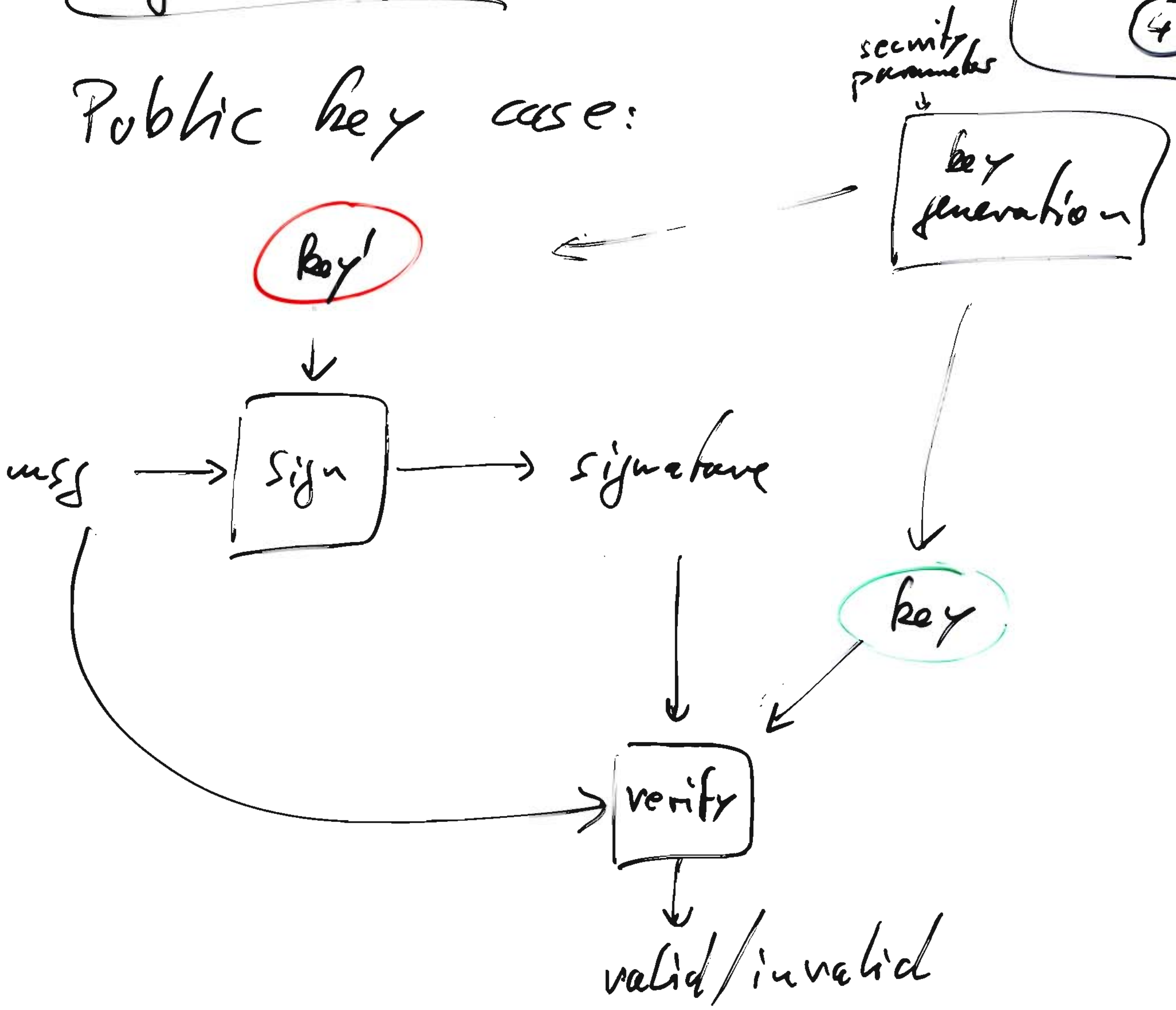
Mostly used as hybrid.

- protects against disclosure, grants confidentiality: only owners of key' can decrypt (if ... ~~the~~ scheme is 'secure' and the problem it is based on is not broken).
- No protection against changes.

esec
14.04.10
4

2 Signatures

Public key case:



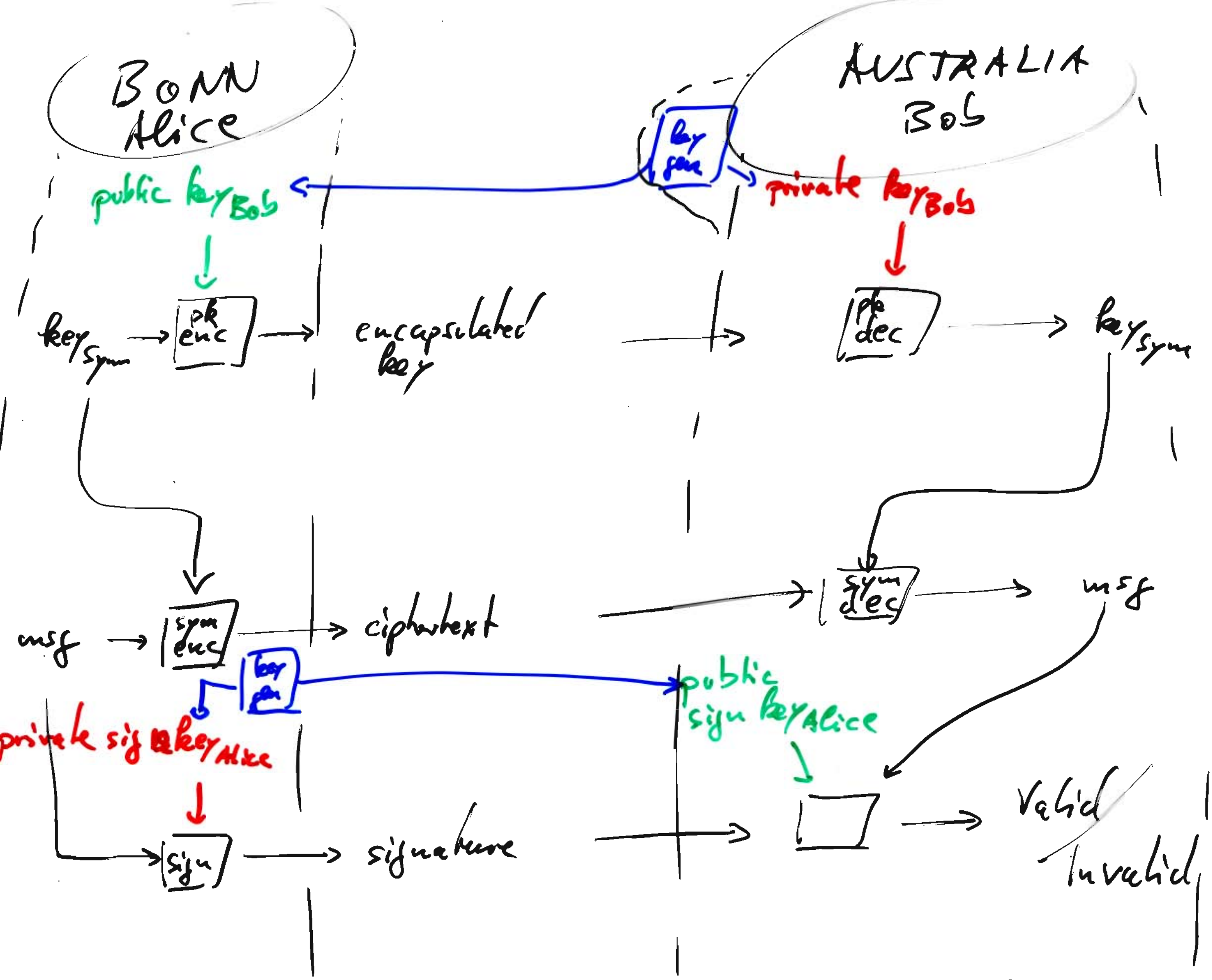
- protect against fake sender (identify)
  (or the scheme is broken)
- protect against manipulation (integrity)
- protect against repudiation (non-repudiation, authenticate)

Symmetric case:
Message Authentication Code.

t.b.done: 3 PKI

Hybrid & authentic message transport

esec 20.04.10



How does Alice know that what she gets as public key$_{Bob}$ actually belongs to Bob? → Confidentiality.

How does Bob know that what he gets as public sign key$_{Alice}$ actually belongs to Alice? → Integrity, Authenticity, Identity of sender.

Need certificates

Identity information
(Name, Picture,
Birthplace, ...)

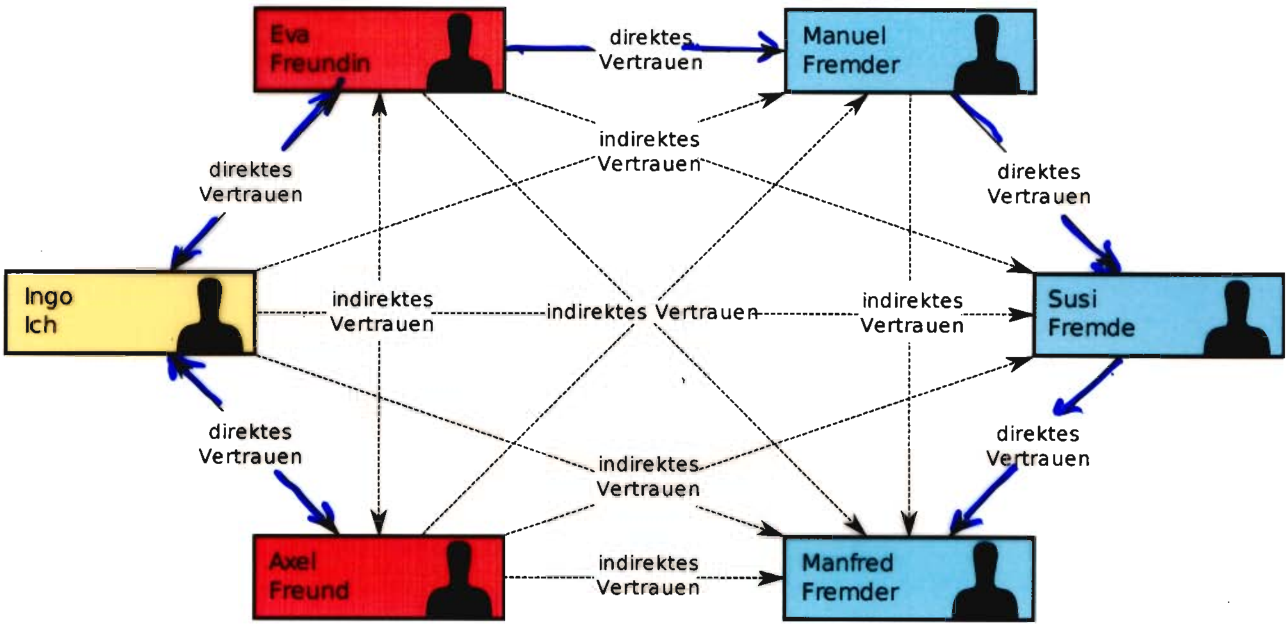
Public key

Public sign key

- signature of a trusted third party

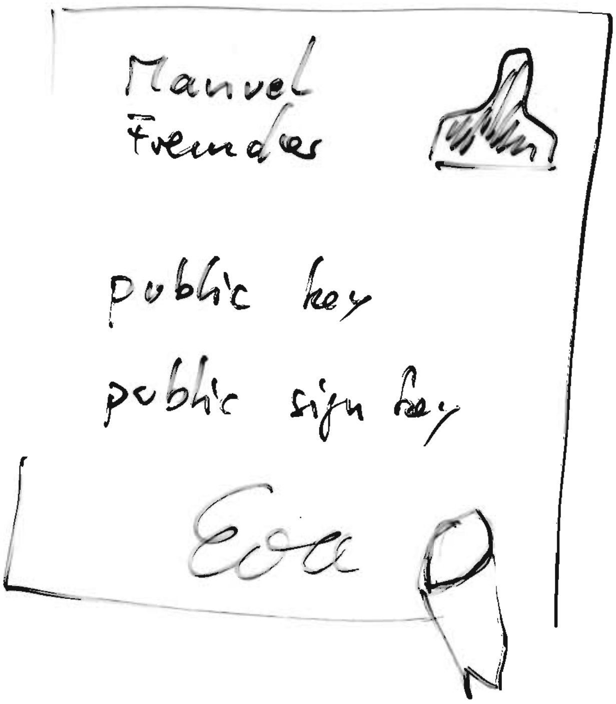
Who signs and how do I know that the signer is actually who I suppose him to be?

2



Web of trust

For example: there is a certificate



→ Solution of PGP, GnuPG
OpenPGP standard

esec
20.4.10
3

Interludium:

1991 Phil Zimmermann: Pretty Good Privacy (PGP) open source

Problem with US export restrictions „40 bit"

Later: software is protected by US constitution „free speech".

1995 Printed source code of PGP in book

1997 Zimmermann sold PGP to McAfee closed source

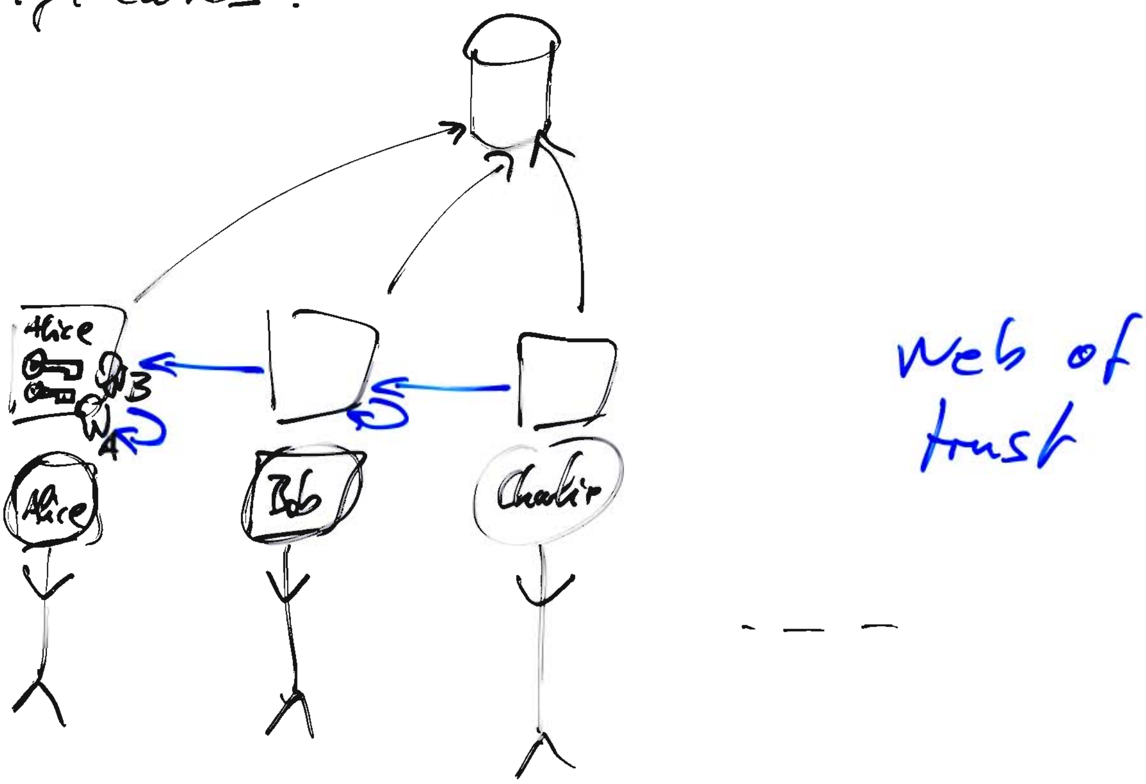
1998 Open PGP standard & GnuPG (open source)

2002 McAfee sold PGP to PGP Corporation who continued the development again in open source.

csec
20.4.10
4

PGP, GnuPG use a keyring. Such a keyring contains several certificates, and any user can sign and so attribute to any id-key relation he likes.

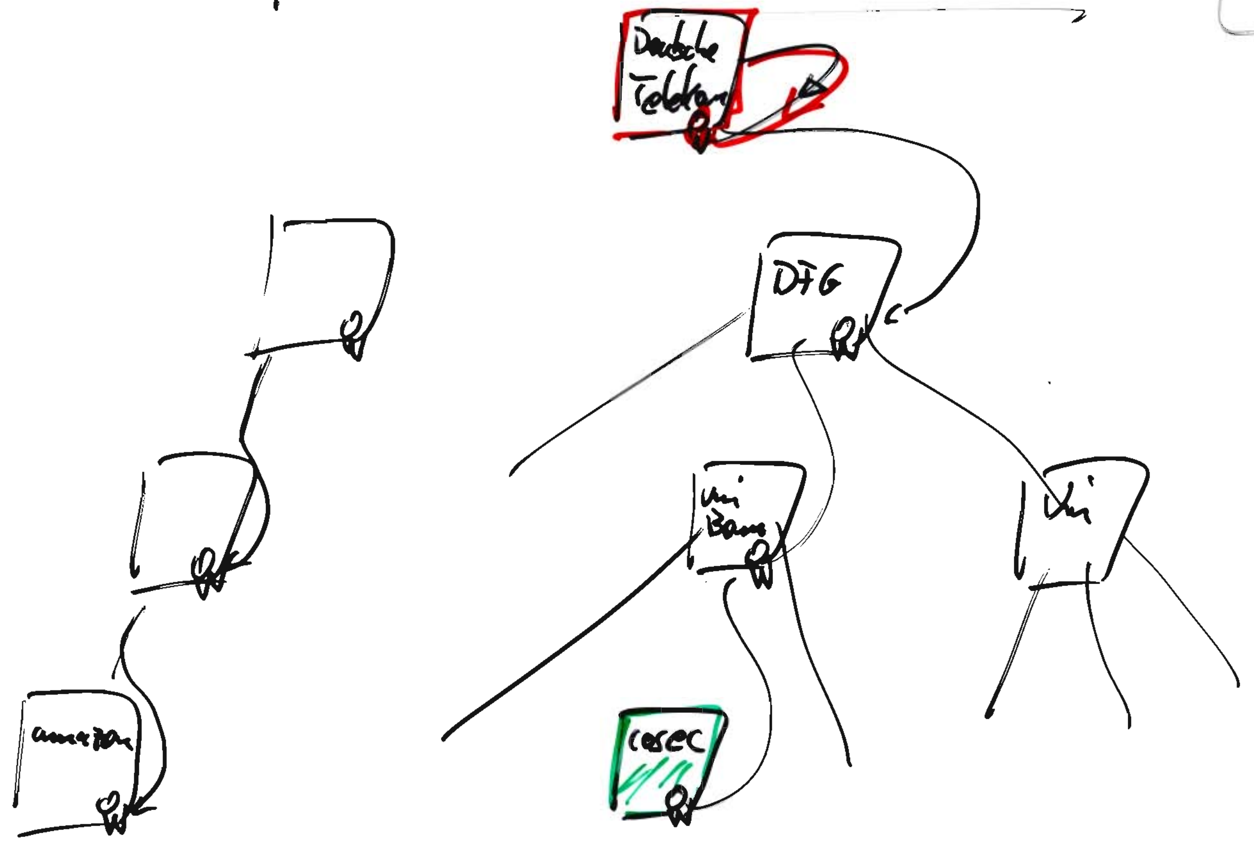
Additionally there are key servers which simply collect lots of these certificates.



Other solutions?

esec
21.4.10
1

Another solution for distributing certificates



hierarchical PKI.

All the trust is anchored in the root certification authority certificates.

You decide which ones are genuine.

The security of such a combined mechanism relies on
- trust in the Root CA certificates
- trust in the CAs
- security of all components (encryption, signatures)

What is security?

- A system is secure if you have to spend more money to break it than the benefit you have from a break.
- A system is secure if the time for breaking it is longer than the lifetime of the attacker.
- A family of systems with ~~a possibly~~ an arbitrarily large security parameter k is secure if the attack complexity (least possible runtime) is not bounded by any polynomial in k.

esec
21.4.10
2

AND

- security of their combination.

Example

RSA: at least have 1024 bit keys: $N = p \cdot q$, (80 bit security ?)

$$e \cdot d \equiv_{(p-1)(q-1)} 1.$$

public key (N, e)

private key (N, d)

$|N|, |e|, |d| \sim 1024$ bits.

message $x \in \mathbb{Z} \cap [0, N-1[$.

~ 1024 bits.

AES: 256 bit key x (best key size)

$$y := \mathrm{enc}_{RSA}(x) = x^e \text{ rem } N$$ (200 bit security ?)

Note: x is always short, only 25% of the possible length.

There mechanisms to extract x from (y, N, e) within seconds. (↑ Lattices) (NO bit security)

esec
27.04.10
1

What attackers do we consider?

- It depends on whether we put on our asymptotic glasses or our fixed size glasses.

in the asymptotic view we always restrict the attacker to

polynomial time

(and polynomial space)

in the fixed size view things are more complicated: for example for 80-bit security we allow 2^{80} run time. (Beware of the time unit!)

„resources of the attacker"

esec
27.04.10
1

What attackers do we consider?

- It depends on whether we put on ~~our~~ asymptotic glasses or our fixed size glasses.

in the asymptotic view we always restrict the attacker to

polynomial time

(and polynomial space)

in the fixed size view things are more complicated: for example for 80-bit security we allow 2^{80} run time.
(Beware of the time unit!)

"resources of the attacker"

- What are its inputs?

Problem specific issue!

Example: RSA.

- Attacker1 gets input N, e, y.
- Attacker2 gets input: N, e, y, power trace of a computation using the secret exponent d.

As ~~far as we know~~
we do not know an attacker1 which is successful.
We do know a attacker2 which is successful!

- What is the aim of the attacker?

For example:

- compute the plain text of an enciphered message
- decide whether a specific word is in the plain text of an enciphered message.
- compute the private key.

And it is enough to be successful "sometimes".

csec
27.04.10
3

~~We want that~~
We call an attacker successful
if he gives a correct answer
with a non-negligibly higher
probability
than a randomly guessing algorithm.

Example | Input: N, e, y.
Output: least significant bit of x
where $y \equiv_N x^e$.

If A has a success probability of 60%
then calling A 3-times and taking
the majority gives us a success
probability of 64% ($>60\%$).

Asymptotic classes:
Amplification: ~~Repeat~~ Majorizing over n executions of A
gives a success probability $\cancel{1 - c \cdot 2^{-n}}$ $\boxed{1 - c \cdot a^{-n}}$
with $c, a > 1$.

if success prob of A is 60%.
Actually, even $50\% + \frac{1}{n^{17}}$ is enough

esec
27.04.10
4

An example

RSA-sign: algorithm designed to fit:

$$\text{RSA-verify}(m, s, \underbrace{(N,e)}_{\text{public RSA key}}) = \boxed{\operatorname{hash}(m) \equiv_N s^e} \to \begin{matrix}\text{TRUE}\\ \text{FALSE}\end{matrix}$$

hash: some function with nice properties outputting $\log_2 N$ bits.

Is this "secure"?

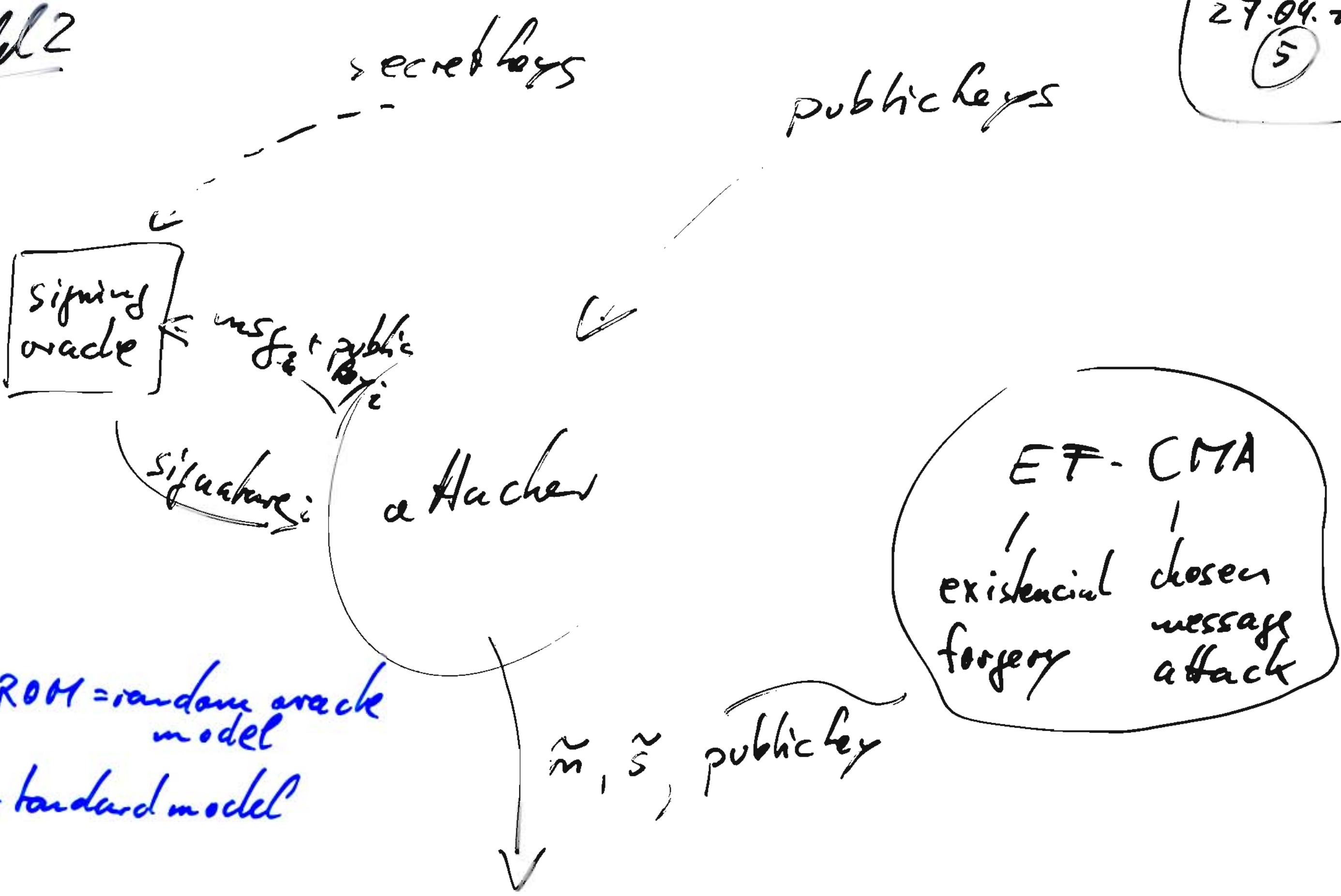
In which sense do we want to ask this question?

Model 1

A successful attacker shall output a forged message-signature pair, ~~(Message Attack)~~ given the public key of the sender, a list of earlier signed messages.

Model 2

esec 27.04.10 5



in ROM = random oracle model

in standard model

The attacker is successful if

$\tilde{m}$, $\widetilde{\text{publickey}}$ was never asked to the oracle.

and $\tilde{m}$, $\tilde{s}$ is a valid msg-signature pair corresponding to $\widetilde{\text{publickey}}$.

and $\widetilde{\text{publickey}}$ $\in$ publickeys.

with a success probability suitably larger than guessing.

Precisely: runtime $\in$ poly, $\frac{1}{\text{successprob}} \in$ poly.

es ec
27.04.10
6

Assume that in RSA-verify the function hash is just the identity.

RSA-verify

$\boxed{\text{hash}(m) \equiv_N s^e}$

Attacker$_1$: output 0, 0, publickey.

Attacker$_2$: choose $s \in_R \cancel{\mathbb{Z}_N}\ \mathbb{N}_{<N}$

compute $m = s^e \text{ rem } N$.

~~output~~ output $m, s, \underline{(N,e)}$

Attacker$_3$:
- fix (N,e) and a message m.
- call the signing oracle twice:

  for $k \cdot m \rightarrow s_1$

  for $k^{-1} \rightarrow s_2$.

- compute $s := s_1 \cdot s_2 \text{ rem } N$
- output $m, s, (N,e)$.

Each of the three attackers proves that RSA-signature with hash = id is

$\underline{\text{NOT}}$ secure.

esec
28.4.10
1

A function $h^{(k)}: \{0,1\}^* \longrightarrow \{0,1\}^k$.
is called collision-resistant
ie. there is no algorithm
that outputs $x_1, x_2 \in \{0,1\}^*$ such that

$$\circ \quad x_1 \neq x_2 \quad \wedge \quad h^{(k)}(x_1) = h^{(k)}(x_2)$$

and expected polynomial time w.r.t. k.

Stupid solution: Input: k
Output: x_1, x_2
1. x_1 is chosen randomly
2. x_2 is chosen randomly,
say both with $2k^2$ bits.
3. Return x_1, x_2.

Repeat this until you find collision.

$$\text{Expected Runtime} = \frac{1}{\text{exit probability}} = 2^k.$$

$$\text{exitprob} = \text{prob}(h^{(k)}(x_1) = h^{(k)}(x_2) \ldots)$$

$$\approx \text{prob}(k \text{ random bits} = k \text{ other random bits})$$

$$= 2^{-k}.$$

esec
28.4.10
2

Better trivial solution
Input: k
Output: x_1, x_2 bitstrings
collision for h.

↑ Birthday attack

1. [Pick $x_1, x_2, x_3, \ldots$
2. Until $\exists i,j : x_i \neq x_j \wedge h^k(x_i) = h^k(x_j)$
3. Return x_1, x_2.

Expected runtime: $O(\sqrt{2^k}) = O(2^{k/2})$

Thus if SHA1, which is a hash function outputting 160 bits,

is as secure as possible ~~the~~

then it would offer 80-bit security.

(Since the above trivial attack runs in time 2^{80} executions of SHA1.)

Side remark: there are attacks on SHA1 which claim run time 2^{63}.

Thus we consider (the collision-resistance of) SHA1 broken.

esec
28.4.10
2

A function $h^{(k)} : \{0,1\}^* \to \{0,1\}^k$
is called one-way
if. it can be computed in polynomial time
and
there is no algorithm
that ~~outputs~~ given a possible hash value $y \in \{0,1\}^k$
outputs $x \in \{0,1\}^*$ such that
$$h^{(k)}(x) = y$$
and in expected polynomial time w.r.t. k = length of y.

Assume that you sign messages
by (1) compute the hash value $y = \text{hash}(m)$
(2) do some specific computation
with y and a secret key...

Claim
If this scheme is (EF-CMA) secure
then the hash function is collision-resistent.

Proof We have to show that
if there is an algorithm for computing collisions
in (expected) poly. time
Then there is an attacker to the EF-CMA-security.

csec
28.4.10
3

Attacker:

1. Call the algorithm for computing a collision: x_1, x_2
2. Call the signing oracle for x_1 and a pk of your choice: s_1.
3. Output $(x_2, s_1, pk.)$

This runs in poly time and always outputs a valid, non-queried message-signature pair whenever the collision-algorithm was successful.

Thus the scheme is not secure in contradiction to the assumption. ◻

Claim 2

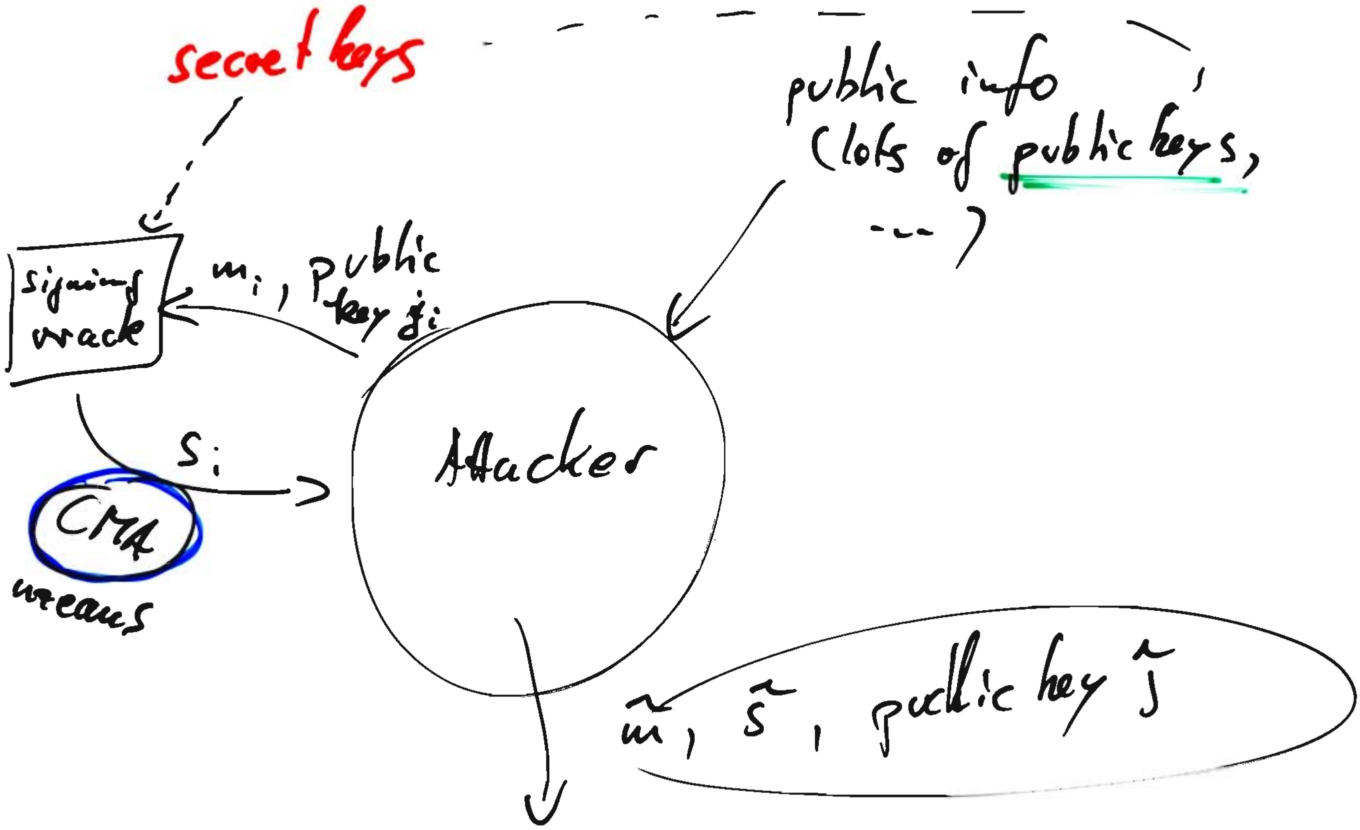
If the signature scheme is (EF-CMA) secure then the hash function is one-way

Pf: Exercise. △

Bottom line: The security model allows us to derive necessary conditions.

4.5.10
esec
1

Model for security of a signature scheme



The attacker is successful if

(1) it produces a validly signed message $\hat{m}$, $\hat{s}$ for user $\hat{j}$ that was never queried

EUF
task

(2) within expected poly time (with small error)

or (2') within poly time with

$$succ = prob(①) > \frac{1}{n^a}$$

for some a and large n.

The scheme is EUF-CMA secure in the standard model if there is no such attacker.

4.5.10
esec
2

A weaker model:

public info
(lots of public keys,
...)

NO
signing
oracle

key only
attack
KOA

secret key to one
of the input public keys

UB Unbreak-
ability

in the RANDOM ORACLE MODEL
(ROM)

that is: think of the hash
function as a random function.

E.g. you could define:

my random fn: Input: $m \in \{0,1\}^*$

Output: $h \in \{0,1\}^{160}$

1. was m queried before?
2. If yes: answer the same h.
3. otherwise choose $h \in_R \{0,1\}^{160}$
4. and put (m,h) into some table
Return (h)

esec
4.5.10
3

For example one has proved:

Theorem

RSA-signatures with a
full domain hash function
is ~~EU~~ existentially unforgeable,
under chosen-message attack
in the ~~R~~random oracle model.
under certain number theory assumption.

EUF
CMA
secure
in ROM.

Want:

RSA-FDH
is EUF-CMA secure
in the standard model
under suitable hypothesis.

BAD points

There exists a constructed scheme
that is secure in ROM.
but with every specific function
in place of the oracle it is
INSECURE.

Connections between different models?

strongest security notion.

$$
\begin{tikzcd}
\text{existential unforgeability } EUF \arrow[d, Rightarrow] & \circ \arrow[d, Rightarrow] & \circ \arrow[l, Rightarrow] \arrow[d, Rightarrow] & \bigcirc \arrow[l, Rightarrow] \arrow[d, Rightarrow] \\
\text{universal unforgeability } UUF \arrow[d, Rightarrow] & \circ \arrow[d, Rightarrow] & \circ \arrow[l, Rightarrow] \arrow[d, Rightarrow] & \circ \arrow[l, Rightarrow] \arrow[d, Rightarrow] \\
\text{un-breakability } UB & \circ & \circ \arrow[l, Rightarrow] & \circ \arrow[l, Rightarrow] \\
& KOA & NACMA \arrow[l, Rightarrow] & CMA \arrow[l, Rightarrow]
\end{tikzcd}
$$

KOA: key-only attack

NACMA: non-adaptive chosen message attack

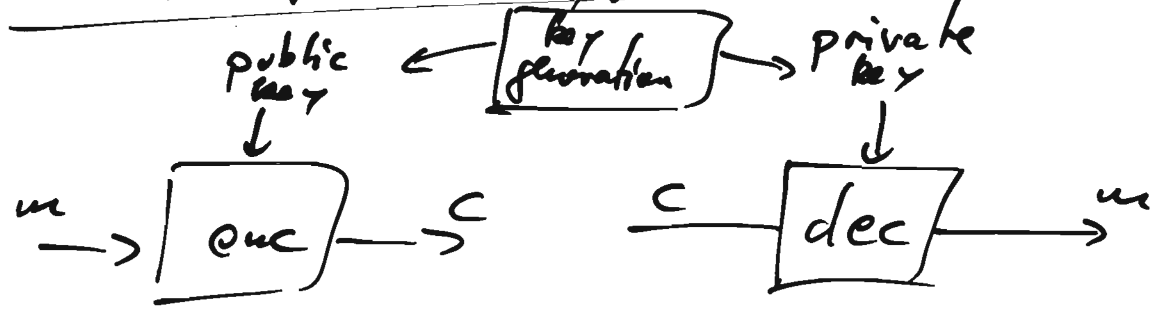
CMA: chosen message attack

security in standard model $\Rightarrow$ security in the random oracle modal

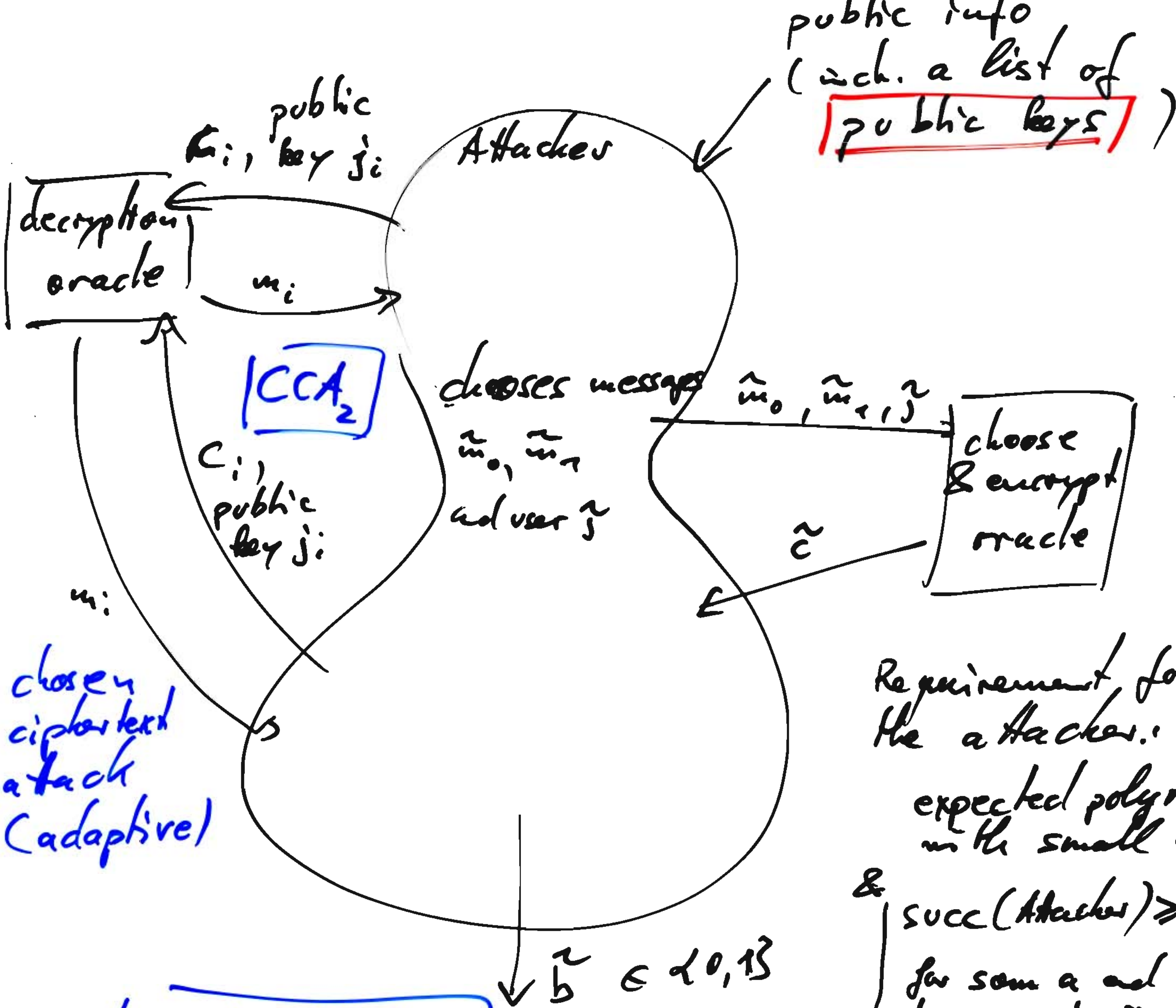
security in standard model $\not\Leftarrow$ security in the random oracle modal

Security for encryption schemes

esec
4.5.10
5



g. RSA: keygen $\to$ (N,e), (N,d) , $enc_{(N,e)}(m) = m^e \text{ rem } N$



Requirement for the attacker:
expected polytime with small error
& succ(Attacker) $\geq \frac{1}{2} + \frac{1}{n^a}$
for some a and large input size n

INDistinguishability

Task: output $\tilde{b}$ such that $dec_{\tilde{j}}\, \tilde{c} = \tilde{m}_{\tilde{b}}$.

Theorem RSA is insecure in this model.

esec
5.5.10
1

Note that

$IND-CCA_2 \overset{!}{\Longleftrightarrow} NM-CCA_2$

Indistinguishability

Non malleability
(This models whether an attacker can intentionally manipulate the plaintext on the encrypted message.)

Note that any <u>deterministic</u> encryption schemes, i.e. where the encryption is a function of public key and message, is <u>NOT</u> $IND-CCA_2$ secure.

Pf Attacker does this:

1. Choose $\hat{m}_0, \hat{m}_1$ arbitrary but different.
2. Send to the encryption oracle and obtain $\hat{c}$ as an encryption of one of the messages.
3. $\hat{c}_0 = enc(\hat{m}_0)$, $\hat{c}_1 = enc(\hat{m}_1)$
   If $\hat{c}_0 = \hat{c}$ then $\hat{b} := 0$,
   If $\hat{c}_1 = \hat{c}$ then $\hat{b} := 1$.
4. Return $\hat{b}$.

Always successful!

□

esec
5.5.10
2

Note Even the ElGamal encryption (in schoolbook variant) is not IND-CCA$_2$-secure, even though it is randomized.

Interlude ElGamal encryption

key generation:
Fix a group $(G, +)$ with a generator $P \in G$ of order ℓ.
Pick $\alpha \in_R \mathbb{Z}_\ell$ (unpredictably).
Compute $A := \alpha P$ in the group G.
Output: private key α,
public key A.

encrypt:
Input: ~~private~~ public key A, message M
Output: ciphertext (Q, C)
1. Pick $\tau \in_R \mathbb{Z}_\ell$ (unpredictably).
2. Compute $Q := \tau P$,
$C := M + \tau A$.
3. Return (Q, C)

decrypt: Input: ciphertext (Q, C), private key α
Output: message M.
1. Return $C - \alpha Q$.

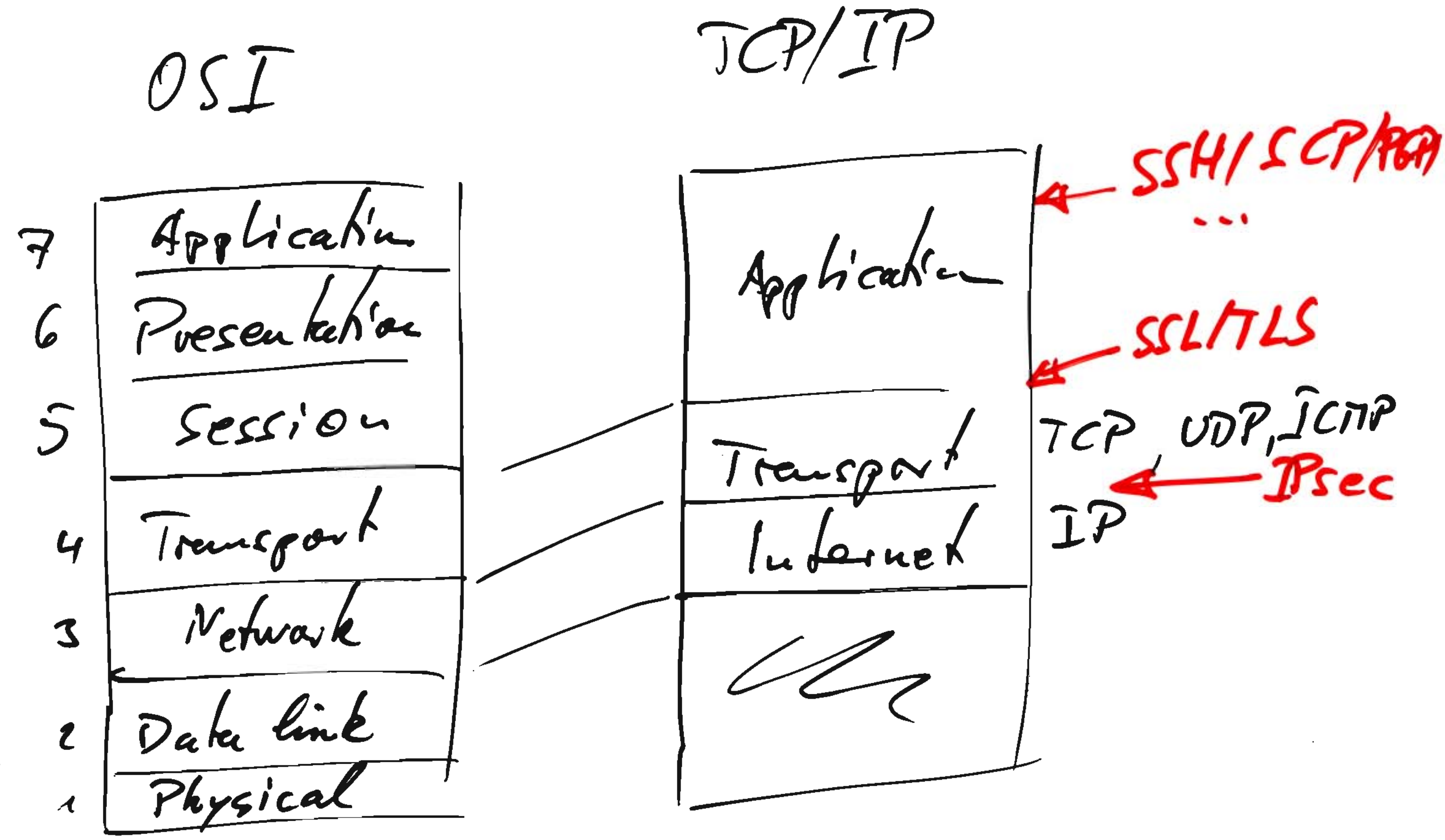
Note that $\mathrm{dec}(Q + \sigma P, C + \sigma A) = \mathrm{dec}(Q, C)$.
This breaks INDistinguishability under CCA$_{1/2}$.

But if you change the combination of M and τA into C
then there is hope to get a
secure scheme.

esec
5.5.10
3

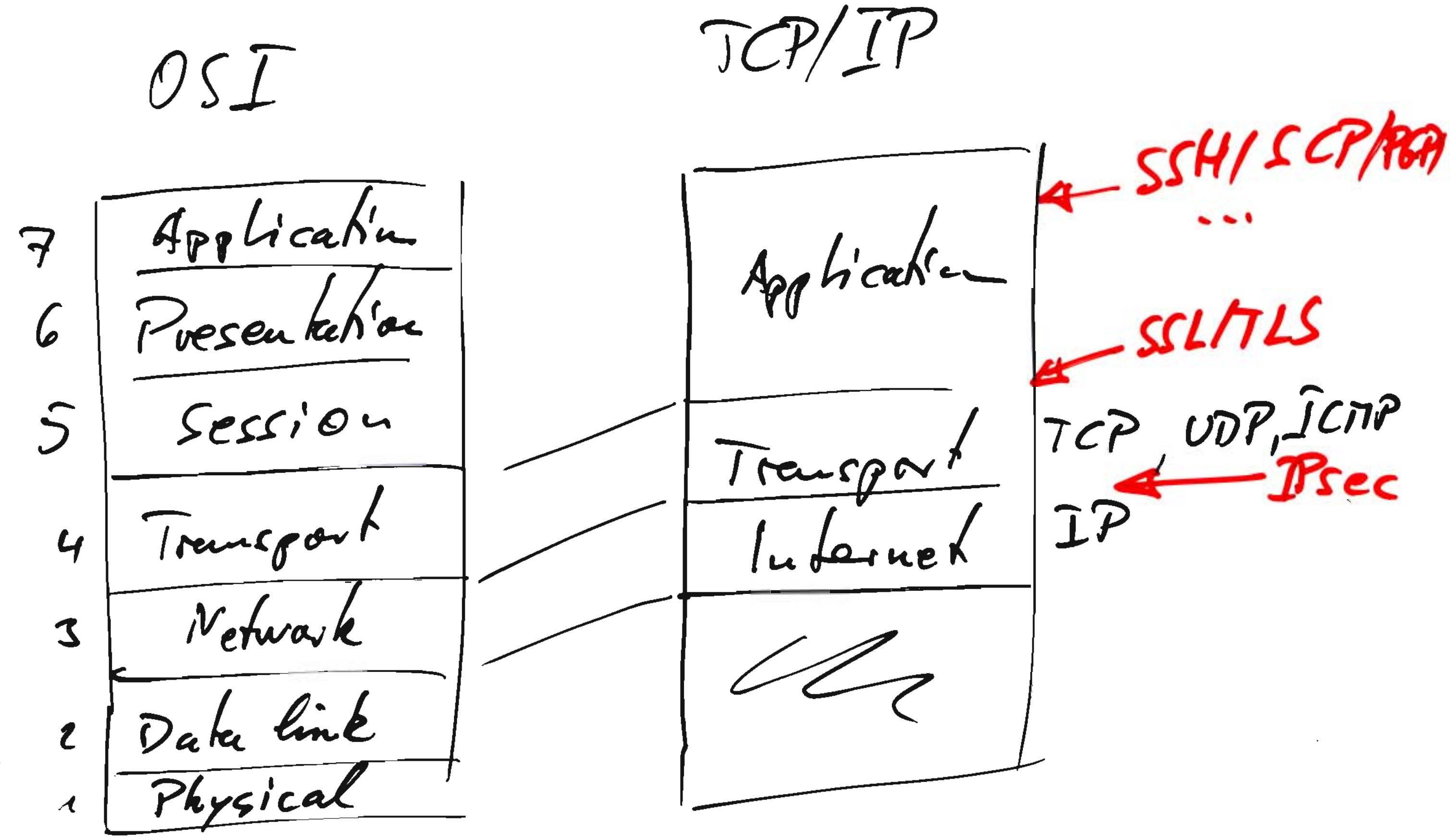
Real world protocols

esec
S.S. 10
4



Real world protocols

esec
S.S. 10
4



IPSEC & IKE

Michael Nüsken

25 June 2007

Before all: we are talking about a collection of protocols. Each partner of the exchange has to keep some information on the connection. This is in our context called the security association (SA). It contains specification about the algorithms that should be used for encryption and authentication, it contains keys for these, it may contain traffic selectors (filtering rules), and more. Each SA manages a simplex connection for one type of service. In each direction there will be an SA for the key exchange (IKE_SA) and one for the encapsulating security payload or for the authentication header. So each partner has to maintain at least four SAs. Such an SA is selected by an identifier, the so-called security parameter index (SPI). It is chosen randomly but so that it is unique.

1. IPsec

The secure internet protocol modifies the internet protocol slightly. We have the choice between transport and tunnel mode. In tunnel mode, an IP packet

IP header	IP payload

is wrapped in with a new IP header and an IPsec header to

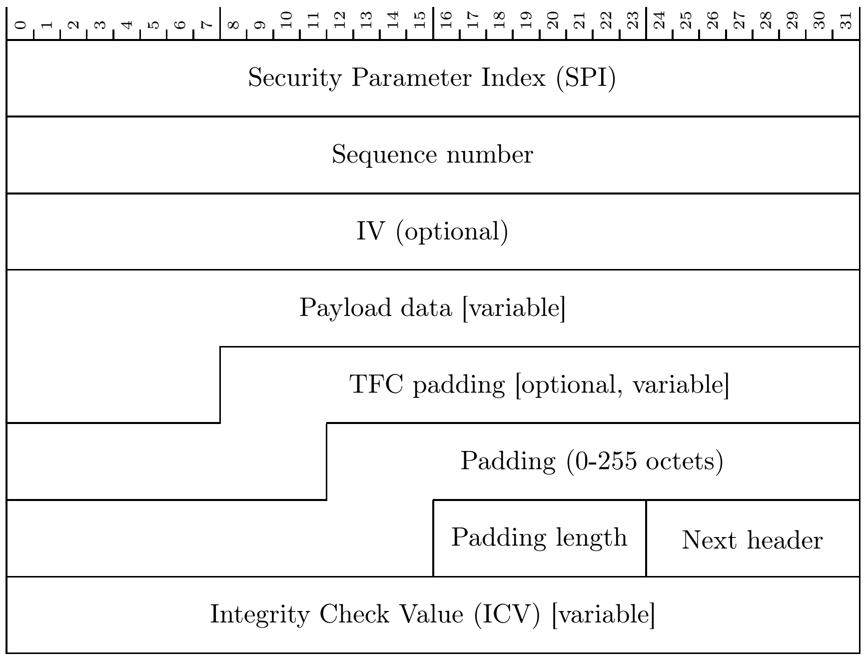
new IP header	IPsec header	IP header	IP payload

In transport mode, only the IPsec header is added:

IP header	IPsec header	IP payload

There are two types of IPsec headers: the encapsulating security payload (ESP) and the authentication header (AH).

1.1. IPsec encapsulating security payload. The ESP specifies that and how its payload is encrypted and (optionally) authenticated. Actually, this 'header' is split into a part before and one after the data:



The security parameter index identifies the SA and thus all necessary algorithms and key material. To create the secured packet from the original one, it is first padded. Padding is used to enlarge the data length to a multiple of a block size that might be associated with the encryption. Traffic flow confidentiality (TFC) padding can be used to disguise the real size of the packet. Then the data is encrypted; in tunnel mode including the old IP header. To be precise, all the information from Payload data to Next header is encrypted. Next, a message authenticion code is calculated for this encrypted text and security parameter index, sequence number, initialization vector (IV) and possibly further padding; actually the message authentication code covers the entire packet but the header and the integrity check value plus the extended sequence number and integrity check padding if any.

1.2. IPsec authentication header. The AH authenticates its payload and also parts of the IP header. (Yes, this does violate the hierarchy.)

2. Internet key exchange (version 2)

Any message in the internet key exchange starts with a header of the form

0–7	8–11	12–15	16–23	24–26	27	28	29	30–31
IKE_SA initiator's SPI								
IKE_SA responder's SPI								
Next payload	Major version	Minor version	Exchange type	X	I	V	R	X
Message ID								
Length								

Clearly, the version is 2.0 with the present drafts (major version: 2, minor version: 0). The flags X are reserved, the I(nitiator) bit is set whenever the message comes from the initiator of the SA, the V(ersion) bit is set if the transmitter can support a higher major version, the R(esponse) bit is set if this message is a response to a message with this Message ID. The header is usually followed by some payloads like

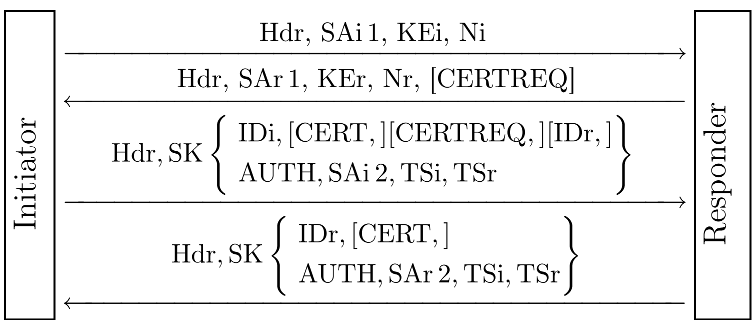
Exchange type	Value
Reserved	0-33
IKE_SA_INIT	34
IKE_AUTH	35
CREATE_CHILD_SA	36
INFORMATIONAL	37
Reserved to IANA	38-239
Reserved for private use	240-255

0–7	8	9–15	16–31
Next payload	C	Reserved(0)	Payload length
Payload			

The C(ritical) bit indicates that the payload is critical. In case the recipient does not support a critical payload it must reject the entire message. A non-critical payload can be simply skipped. All the payloads defined in RFC4306 are to be handled as critical ones whatever the C bit says.

Next payload	Notation	Value
None		0
RESERVED		1-32
Security Association	SA	33
Key Exchange	KE	34
Identification - Initiator	IDi	35
Identification - Responder	IDr	36
Certificate	CERT	37
Certificate Request	CERTREQ	38
Authentication	AUTH	39
Nonce	Ni, Nr	40
Notify	N	41
Delete	D	42
Vendor ID	V	43
Traffic Selector - Initiator	TSi	44
Traffic Selector - Responder	TSr	45
Encrypted	E	46
Configuration	CP	47
Extensible Authentication	EAP	48
Reserved to IANA		49-127
Private use		128-255

2.1. Initial exchange.



Protocol 2.1. IKE_SA_INIT.

1. Prepare SAi1, the four lists of supported cryptographic algorithms for Diffie-Hellman key exchange (groups), for the pseudo random function used to derive keys, for encryption, and for authentication. Guess the group for Diffie-Hellman and compute KEi $= g^a$.
   Choose a nonce Ni. $\xrightarrow{\text{Hdr, SAi 1, KEi, Ni}}$
2. Choose SAr1 from SAi1 unless no variant is supported.

Compute KEr $= g^b$ if the group was guessed correctly. (Otherwise send:

$$\mathrm{Hdr}, \mathrm{N}(\texttt{INVALID\_KE\_PAYLOAD}, \mathrm{group})$$

.)
Choose a nonce Nr.

$\longleftarrow$ $\mathrm{Hdr}, \mathrm{SAr\,1}, \mathrm{KEr}, \mathrm{Nr}, [\mathrm{CERTREQ}]$

3. Both parties now derive the session keys. We assume that prf is the selected pseudo random function which gets a key and a bit string as input.

$$\mathrm{SKEYSEED} = \mathrm{prf}(Ni|Nr, g^{ab}),$$
$$\mathrm{SK\_d|SK\_ai|SK\_ar|SK\_ei|SK\_er|SK\_pi|SK\_pr}$$
$$= \mathrm{prf{+}}(\mathrm{SKEYSEED}, \mathrm{Ni} \,|\, \mathrm{Nr} \,|\, \mathrm{SPIi} \,|\, \mathrm{SPIr})$$

where $\mathrm{prf{+}}(K, S) = T_1|T_2|T_3|\ldots$, and $T_1 = \mathrm{prf}(K, S|0x01)$, $T_i = \mathrm{prf}(K, T_{i-1}|S|i)$ for $i > 1$. SK_d is used for the derivation of keys in a child SA. SK_ai and SK_ei are used for authenticating and encrypting messages sent by the initiator, SK_ar and SK_er for messages sent by the responder.

4. The initiator send its identity IDi, optionally one or more certificates CERT, a certificate request CERTREQ (possibly including a list of trusted CAs), and optionally the responders identity IDr (it may be that the responder serves multiple identities 'behind' it).
Further she computes an authentication AUTH (using the key from the first CERT payload) for the entire first message concatenated with the responder's nonce Nr and the value prf(SK_pi, IDi). The authentication method can be RSA digital signature (1), shard key message integrity code (2), or DSS digital signature (3).

0–7	8	9–15	16–31
Next payload	C	Reserved(0)	Payload length
Auth method	Reserved		
Authentication data			

The initiator starts to negotiate a child SA in SAi 2 with proposed traffic selectors TSi, TSr.

$$\mathrm{Hdr}, \mathrm{SK}\left\{\begin{array}{l}\mathrm{IDi}, [\mathrm{CERT},] \\ [\mathrm{CERTREQ},] \\ [\mathrm{IDr},] \\ \mathrm{AUTH}, \mathrm{SAi\,2}, \\ \mathrm{TSi}, \mathrm{TSr}\end{array}\right\} \longrightarrow$$

5. The responder sends its identity IDr, certificate(s). He computes an authentication AUTH for the entire second message concatenated with the initiator's nonce Ni and the value prf(SK_pr, IDr). Further he supplies the answer SAr 2 to the child SA creation and sends the accepted traffic selectors TSi, TSr.

$$\xleftarrow{\mathrm{Hdr}, \mathrm{SK} \left\{ \begin{array}{l} \mathrm{IDr}, [\mathrm{CERT},] \\ \mathrm{AUTH}, \mathrm{SAr}\,2, \\ \mathrm{TSi}, \mathrm{TSr} \end{array} \right\}}$$

If this initial exchange is completed successfully the IKE_SA and a CHILD_SA are ready for use. Keying material for the childs is generated similar to the IKE_SA keys:

$$\mathrm{KEYMAT} = \mathrm{prf+}(\mathrm{SK\_d}, \mathrm{Ni} \mid \mathrm{Nr})$$

2.2. Creating additional child SAs. Further childs can be created under this IKE_SA using a CREATE_CHILD_SA exhange:

$$\boxed{\text{Initiator}} \quad \begin{array}{c} \xrightarrow{\mathrm{Hdr}, \mathrm{SK} \left\{ \begin{array}{l} [\mathrm{N},]\, \mathrm{SAi}\,2, \mathrm{Ni}, [\mathrm{KEi},] \\ [\mathrm{TSi}, \mathrm{TSr}] \end{array} \right\}} \\ \xleftarrow{\mathrm{Hdr}, \mathrm{SK} \left\{ \begin{array}{l} \mathrm{SAr}\,2, \mathrm{Nr}, [\mathrm{KEr},]] \\ [\mathrm{TSi}, \mathrm{TSr}] \end{array} \right\}} \end{array} \quad \boxed{\text{Responder}}$$

In case a CHILD_SA shall be rekeyed the notification payload N of type REKEY_SA specifies which SA is rekeyed. This can be used to established additional SAs as well as to rekey ages ones. Create new ones and afterwards delete the old ones. Also the IKE_SA can be rekeyed similarly.

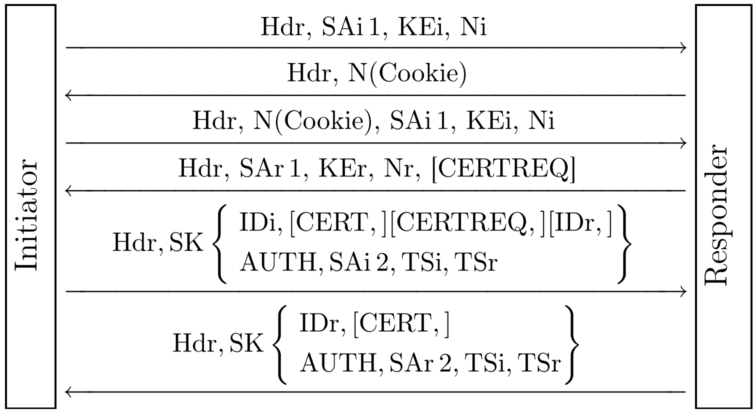
In a CREATE_CHILD_SA exchange including an optional Diffie-Hellman exchange new keying material uses also the new Diffie-Hellman key g^{ir}, it is concatenated left to the nonces. (Though the Diffie-Hellman key exchange is optional, it is recommended to either used it or at least to limit the number of uses of the original key.)

2.3. Denial of Service. If the server has a lot of half open connections (ie. the first message arrived, the second was sent but the third message is pending) it may choose to send a cookie first. (In order to defeat a denial of service attack.) It is suggested to use a stateless cookie consisting of a version identifier and a hash value of the initiator's nonce Ni, her IP IPi, her security parameter index SPIi and some secret:

$$\mathrm{Cookie} = \mathrm{verID} \mid \mathrm{hash}(\mathrm{Ni}, \mathrm{IPi}, \mathrm{SPIi}, \mathrm{secret}_{\mathrm{verID}})$$

This way the secret can be exchanged periodically, say every second, and the server only needs to store the last few (randomly) generated secrets.

The authentication AUTH then refers to the second version of the corresponding message, so the one including the cookie or responding to that, respectively. So the protocol becomes:



2.4. Extended authentication protocols. The initiator may leave out AUTH and thereby tell the responder that she wants to perform an extensible authentication which is then carried out immediately.

2.5. IP compression. The parties can negotiate IP compression.

2.6. ID payload. The ID payload

0 1 2 3 4 5 6 7	8	9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Next payload	C	Reserved(0)	Payload length
ID type	Reserved		
Identification data			

can be an IP address (ID type 1), a fully-qualified domain name string (2), a fully-qualified RFC822 email address string (3), an IPv6 address (5), an ASN.1 X.500 Distinguished Name [X.501] (9), an ASN.1 X.500 general name [X.509] (10), a vendor specific information (11).

2.7. CERT payload. The CERT payload

0 1 2 3 4 5 6 7	8	9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Next payload	C	Reserved(0)	Payload length
Cert encoding	Certificate data		
Certificate data			

can be encoded in various widely used formats. Note that it can also carry revocation lists.

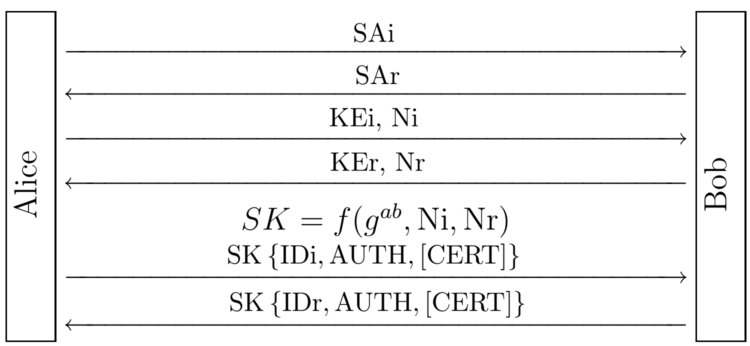
3. IKE version 1

The version 1 of the internet key exchange distinguishes between a main mode and an aggressive mode. Further it allows four variants in each mode depending on the desired type of authentication. Authentication can be based on

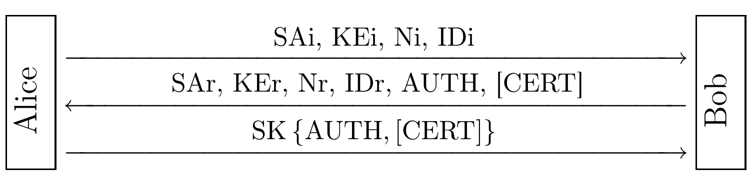
- public signature keys,
- public encryption keys, originial protocol,
- public encryption keys, revised protocol, or
- a pre-shared secret.

We only give the bare protocol summaries here, using notation similar to the one used for version 1. (They are not based on RFC240x but on the book ?.)

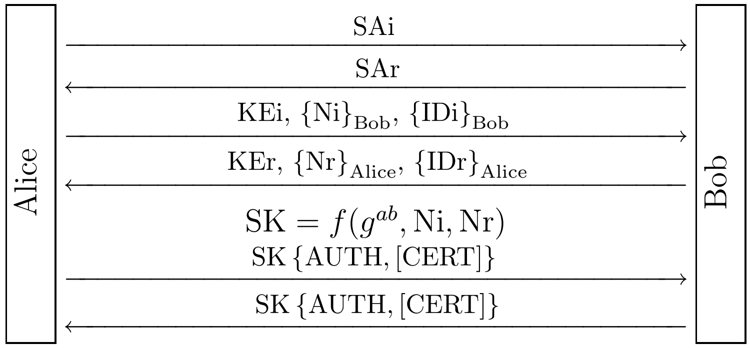
3.1. Main mode, public signature keys.



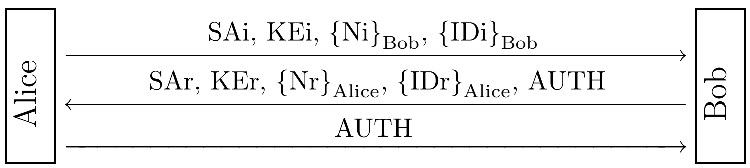
3.2. Aggressive mode, public signature keys.



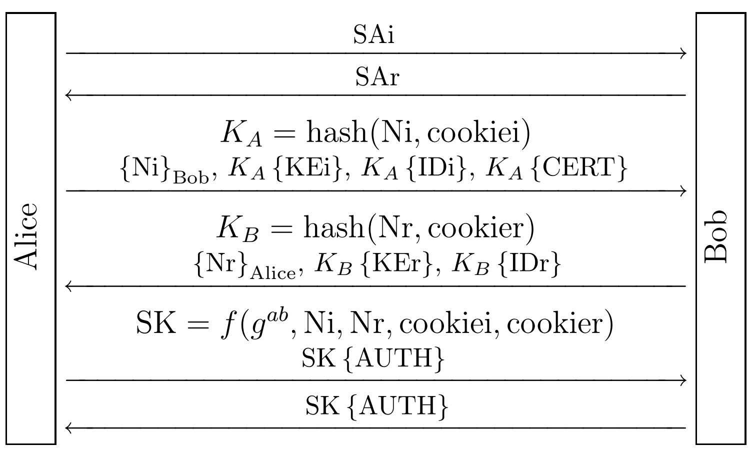
3.3. Main mode, public encryption keys, original protocol.



3.4. Aggressive mode, public encryption keys, original protocol.



3.5. Main mode, public encryption keys, revised protocol.



3.6. Aggressive mode, public encryption keys, original protocol.

Alice | Bob

$$K_A = \mathrm{hash}(\mathrm{Ni}, \mathrm{cookiei})$$

$$\xrightarrow{\text{SAi, }\{\text{Ni}\}_{\text{Bob}}\text{, } K_A\,\{\text{KEi}\}\text{, } K_A\,\{\text{IDi}\}\text{, } K_A\,\{\text{CERT}\}}$$

$$K_B = \mathrm{hash}(\mathrm{Nr}, \mathrm{cookier})$$

$$\xleftarrow{\text{SAr, }\{\text{Nr}\}_{\text{Alice}}\text{, } K_B\,\{\text{KEr}\}\text{, } K_B\,\{\text{IDr}\}\text{, AUTH}}$$

$$\mathrm{SK} = f(g^{ab}, \mathrm{Ni}, \mathrm{Nr}, \mathrm{cookiei}, \mathrm{cookier})$$

$$\xrightarrow{\text{SK}\,\{\text{AUTH}\}}$$

3.7. Main mode, pre-shared secret.

Alice | Bob

$$\xrightarrow{\text{SAi}}$$

$$\xleftarrow{\text{SAr}}$$

$$\xrightarrow{\text{KEi, Ni}}$$

$$\xleftarrow{\text{KEr, Nr}}$$

$$\mathrm{SK} = f(\mathrm{secret}, g^{ab}, \mathrm{Ni}, \mathrm{Nr}, \mathrm{cookiei}, \mathrm{cookier})$$

$$\xrightarrow{\text{SK}\,\{\text{IDi, AUTH}\}}$$

$$\xleftarrow{\text{SK}\,\{\text{IDr, AUTH}\}}$$

3.8. Aggressive mode, pre-shared secret.

Alice | Bob

$$\xrightarrow{\text{SAi, KEi, Ni, IDi}}$$

$$\xleftarrow{\text{SAr, KEr, Nr, IDr, AUTH}}$$

$$\xrightarrow{\text{AUTH}}$$

$$\mathrm{SK} = f(\mathrm{secret}, g^{ab}, \mathrm{Ni}, \mathrm{Nr}, \mathrm{cookiei}, \mathrm{cookier})$$

Michael Nüsken
b-it, Bonn, Germany

IPsec

AH – authentication header

ESP – encapsulating security payload

AH
- ↓ signature
  - ↓ integrity, authenticity

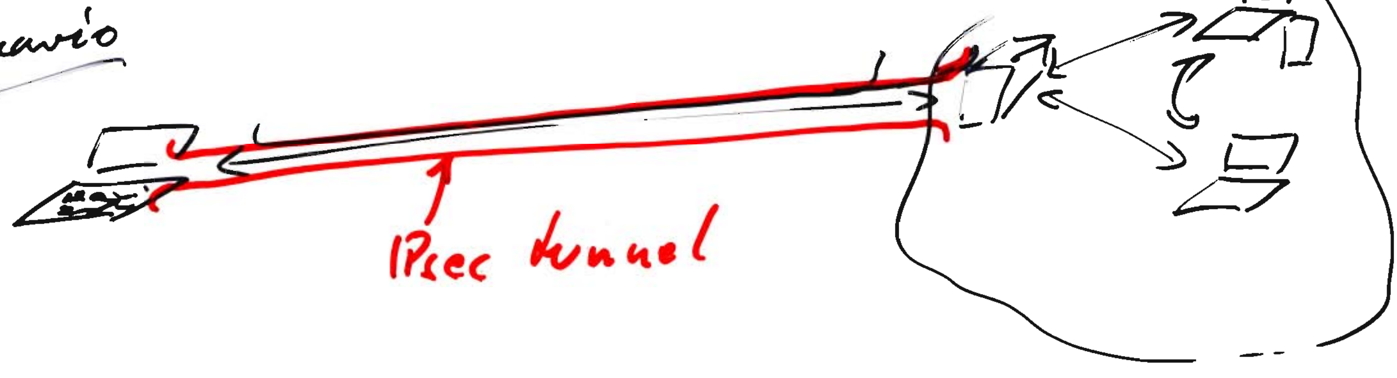
ESP
- ↓ encryption (maybe NULL)
  - ↓ confidentiality
- optionally → signature
  - ↓ integrity, authenticity

Remark AH: authenticates ~~signs~~ some fields of the IP header!

Pro: we would to authenticate that info.

Con: it mixes the hierarchy.

Scenario



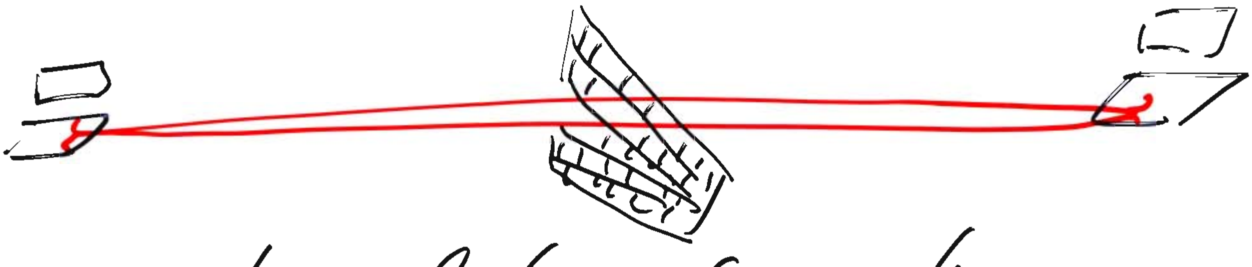
csec
12.5.10
2

Problematic issue

NAT network address translation

(This is basically a workaround to increase the number of addressable devices via IP, which has 'only' 32 bit addresses. This is sorted by IPv6 which has 128 bit addresses and somehow it also has IPsec 'quasi' builtin.)

Firewalls



Encryption hides information about higher level protocol data, like TCP ports.
So the firewall filter packets according to that info.

History of IKE

esec
12.5.10
3

PHOTURIS ⚡ SKIP

NSA proposed ISAKMP

Internet Security Association Key Management Protocol

Not true, only framework.

- only framework
- ruled out both candidates

→ IETF could take up development

→ OAKLEY, SCHEME ... (new drafts)

IKEv1 puts into ISAKMP.

Pro: Something working was there.

Con:
- Development took much too long ... ⇒ IPv6 delayed.
- No clear design
- Too many variants.
- Documentation was awful & difficult to read. ≥ 150 pages, ≥ 3 RFC

esec
12.5.10
4

IKEv2 learned many lessons from that:

- clear, simple rules.
- any request gets a response.
- initial exchange: 1 option (rather than 8), 4 messages
- create child SA: 2 messages
- all functionalities of IKEv1 is still there.

→ easier anlysis

Security questions

esec
12.5.10
5

(0) Secure? → defer this.

(1) Session key agreement

- How long? Random? Unpredictable?

Too small: The impredictability must leave more cases to the attacker than he can try out.

(Linux bug in ~~a~~crypto library caused the pseudo random generator to be initialized ~~with~~ out of $3 \cdot 2^{15}$ cases.)

- Do both parties contribute to it?
- Man in the middle? IPsec: No! secure!

(2) Perfect forward security

- (Beagle boys) Can an attacker decrypt given the long-term secrets after termination of the connection?

Escrow fo(i)tage

- .... during the connection?

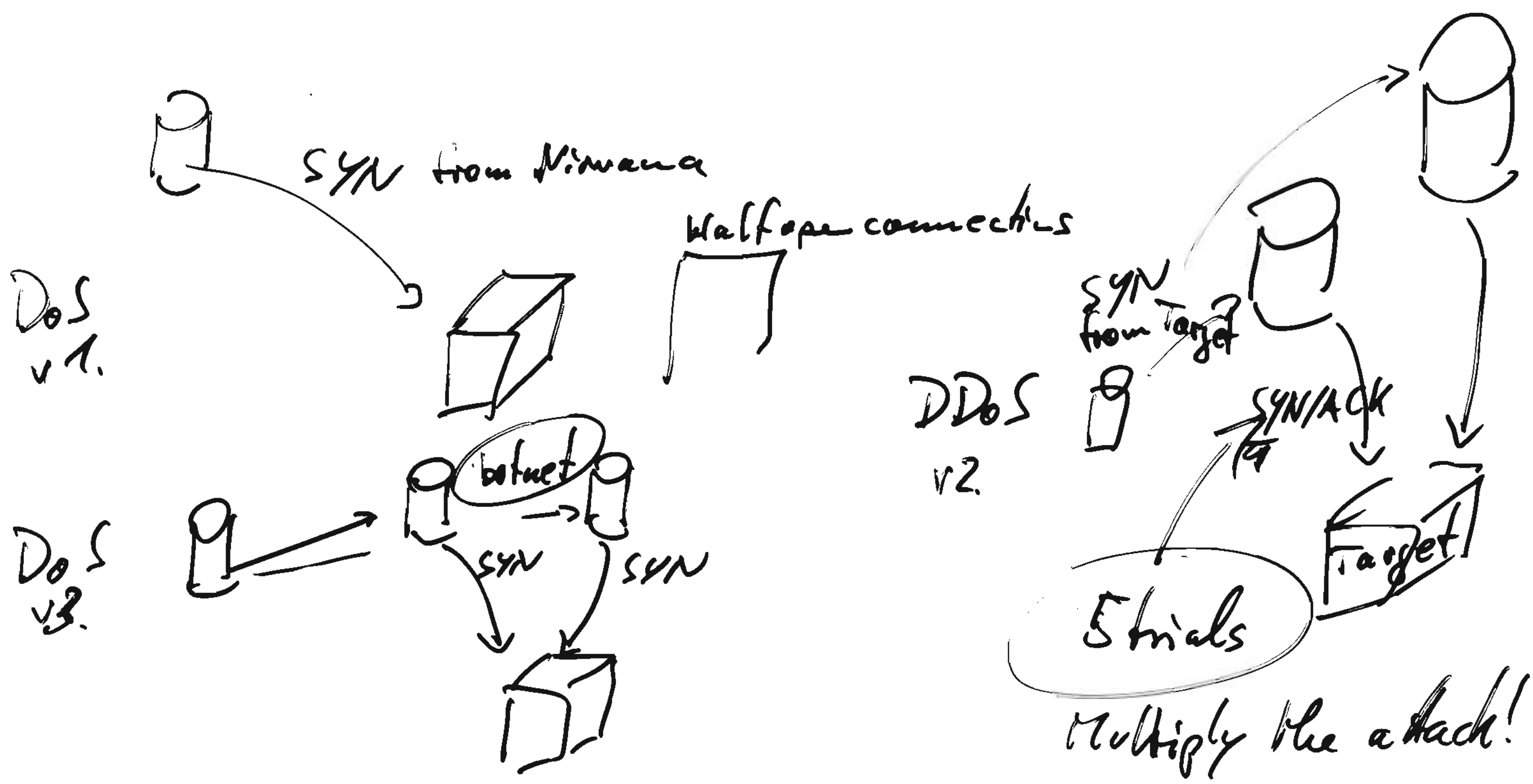
esec
18.5.10
1

News: no class tomorrow.
no classes next week.

→ Time for a project!
See the exercise sheet.

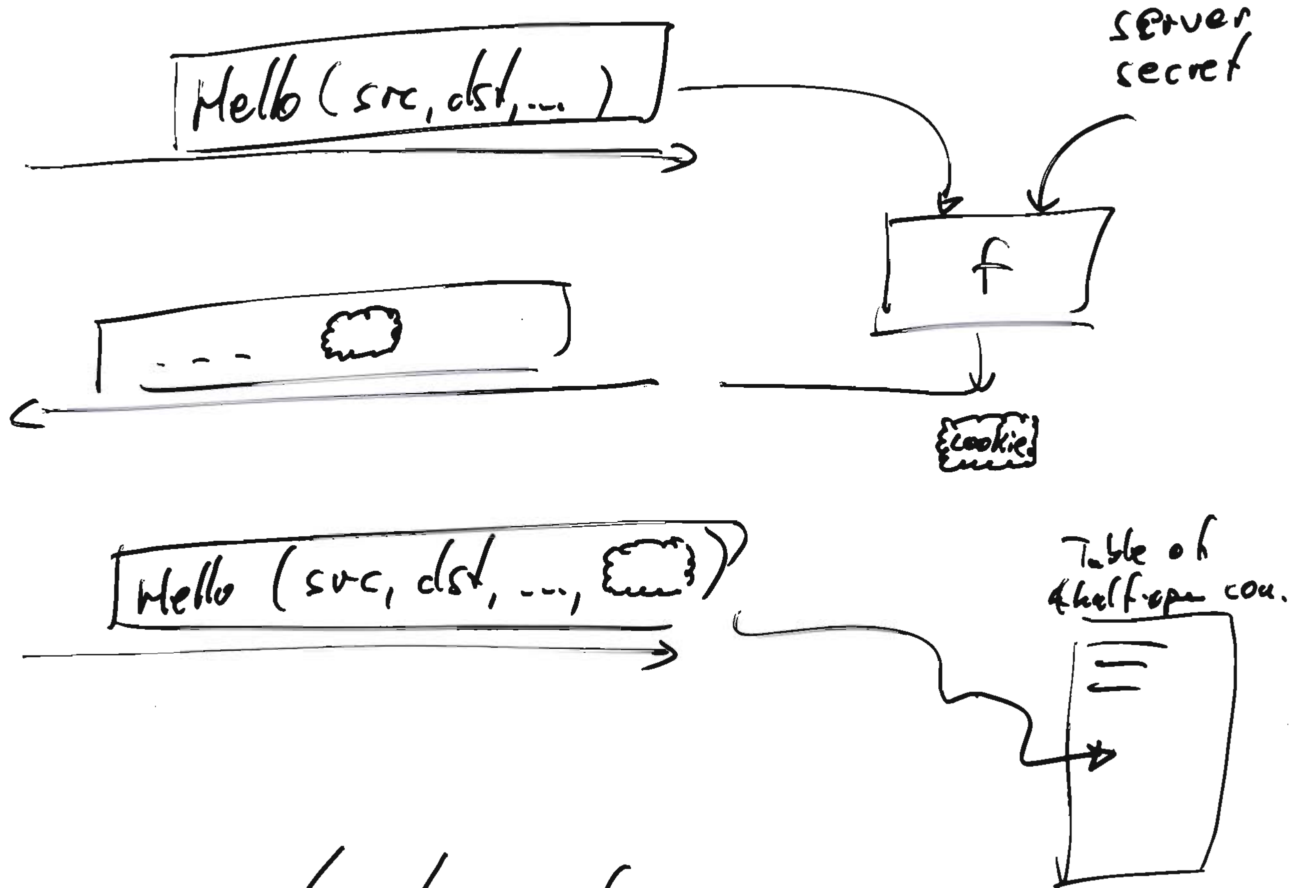
Security questions

0. Secure?
1. Session key agreement.
2. Perfect forward security
   Escrow foilage
3. Denial of Service



~~One~~ solutions

- Increase table sizes.
- Use stateless cookies

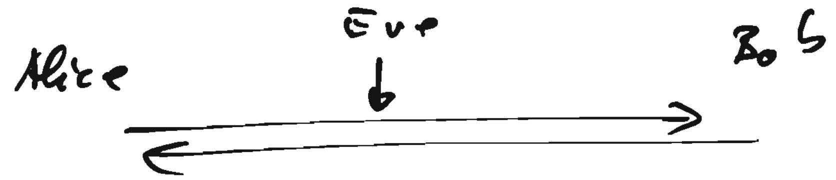


This construct makes the ~~servers~~ (responders) much more resilient DoS attacks.

The cookie must unpredictable!

4 Endpoint identifier hiding

- Does an eavesdropper get information about the identities?



Unless encryption is broke Eve won't be able to see anything.

- Can an active attacker collect identity info?

In IPsec we can protect against revealing the server identity IDr.

Note: It is impossible to protect both identities.

It's a design decision who is protected.

5 Live partner reassurance

Replay attack possible?

→ Protection is done using Nonces (number to be used once)

Use ~~the~~ Nonces to determine the key material.

Then a replay will be not be possible because there ~~is~~ is different session key material.

esec
18.5.10
4

(6) Plausible deniability

Does the protocol log prove that

- Alice talked? → No in IPsec
- Bob talked? → No in IPsec
- Alice or Bob talked? → Yes!
- Alice talked to Bob? } No.
- Bob talked to Alice? }
- One of the last two? Yes.

(7) Stream protection

- How is the logical data stream protected?
  - confidentiality?
    IPsec: Yes (optional!)
  - authenticity?
    IPsec: only partially!
  - integrity?
    IPsec: yes.

(8) Negotiating parameters

esec
18.5.10
5

Pros: flexibility
vs. attacks unknown
at publication time...

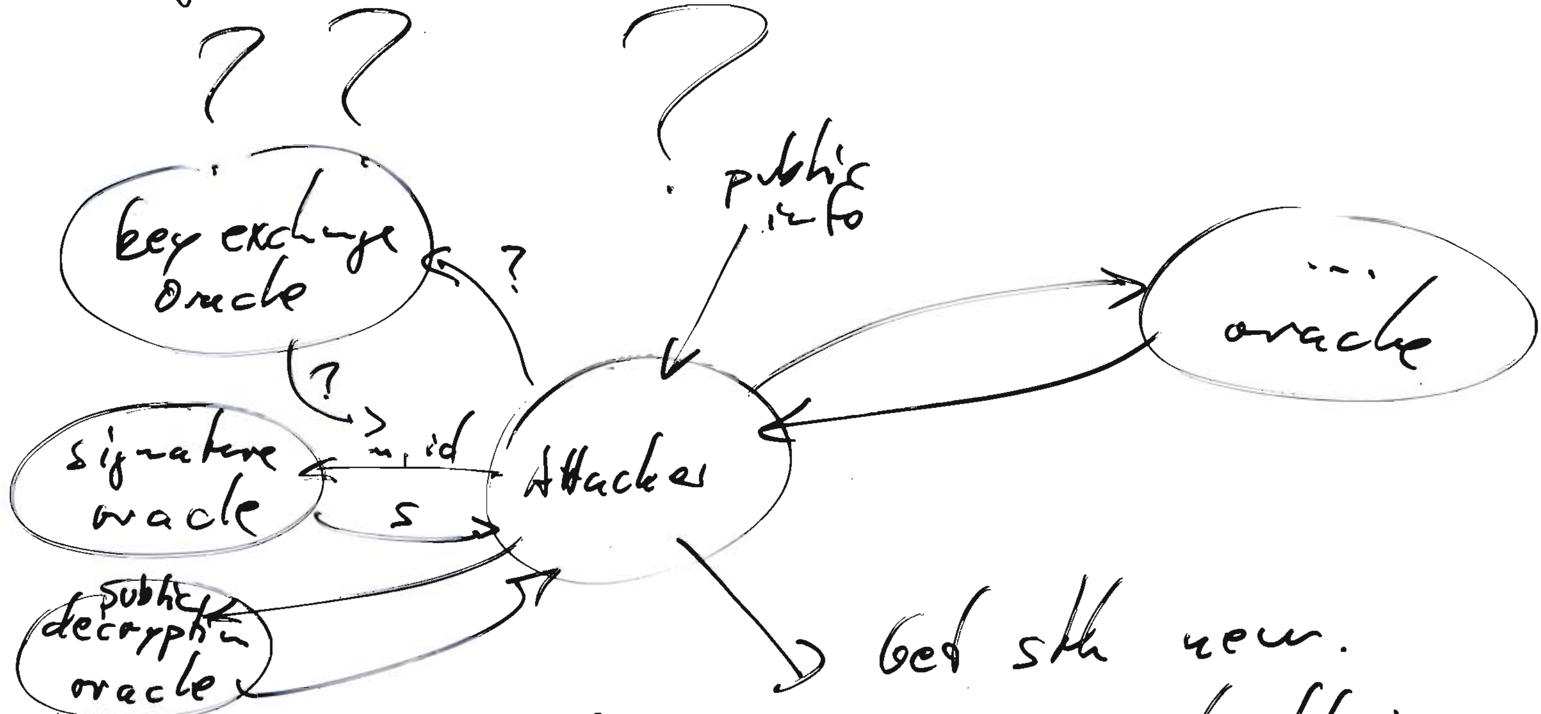
Cons: • System admins might
not know what they do.

• Attackers might use
the negotiation to
"downgrade".

↳ Solved by the V-bit in IPsec
"upgrade"-bit.

1.6.10
1

Def of Secure Protocol



I have never seen such def. in use.

A secure connection

(without a definition of security, at present)

key exchange

Diffie-Hellman

need:

- "secure" group
- good (pseudo) random generator

↑ Debian buggy implementation

authentication + session agreement

need:

- public-key signatures
- PKI (public key infrastructure)

→ social problems

data exchange

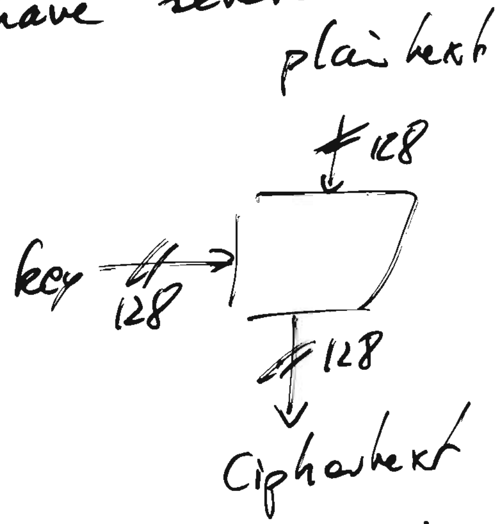
need FAST things!

- fast encryption (⇒ symmetric)
- fast authentication + integrity protection (⇒ symmetric)

- SSL & Discussion
- SSH & Discussion

Fast symmetric encryption

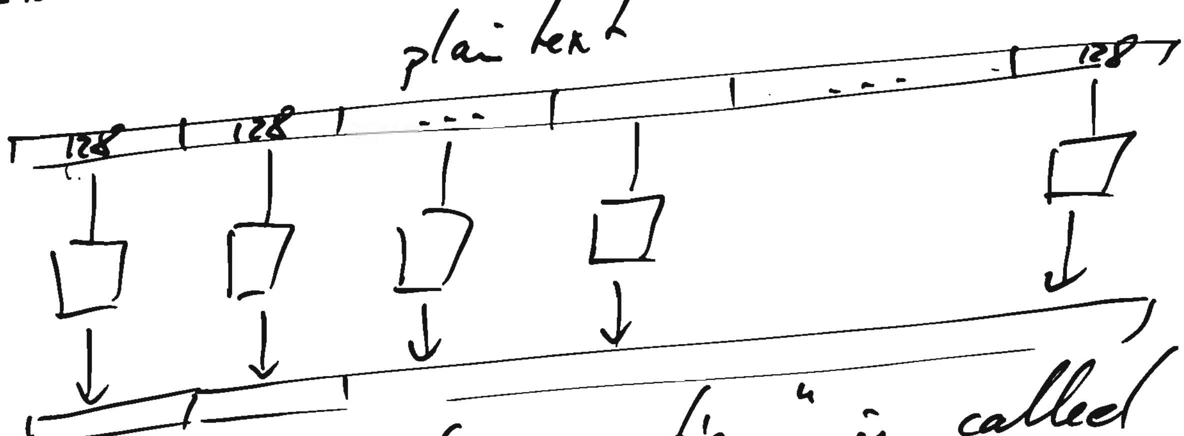
We have several block ciphers, eg. AES:



How to encrypt longer data?

Easiest solution: use it blockwise.

ECB-encryption



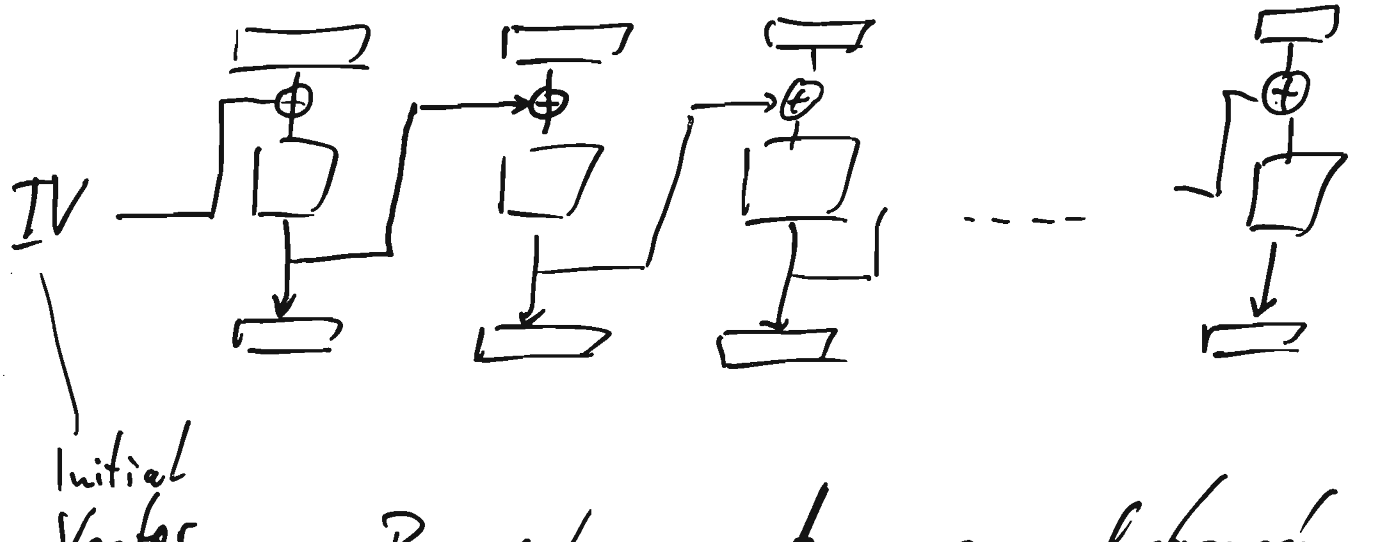
This „mode of operation" is called
Electronic Code Book (ECB).

Problem: Large patterns remain visible.

esec
8.6.10
2

Next best solution: CBC - mode

cipher block chaining



Pro: • Large patterns are destroyed.
• One plaintext has various different cipher text (acc. to the IV).

Con: • if one block is bad more than one blocks are affected.

Pro: • but only two blocks are bad if one block is damaged
→ self-synchronizing.

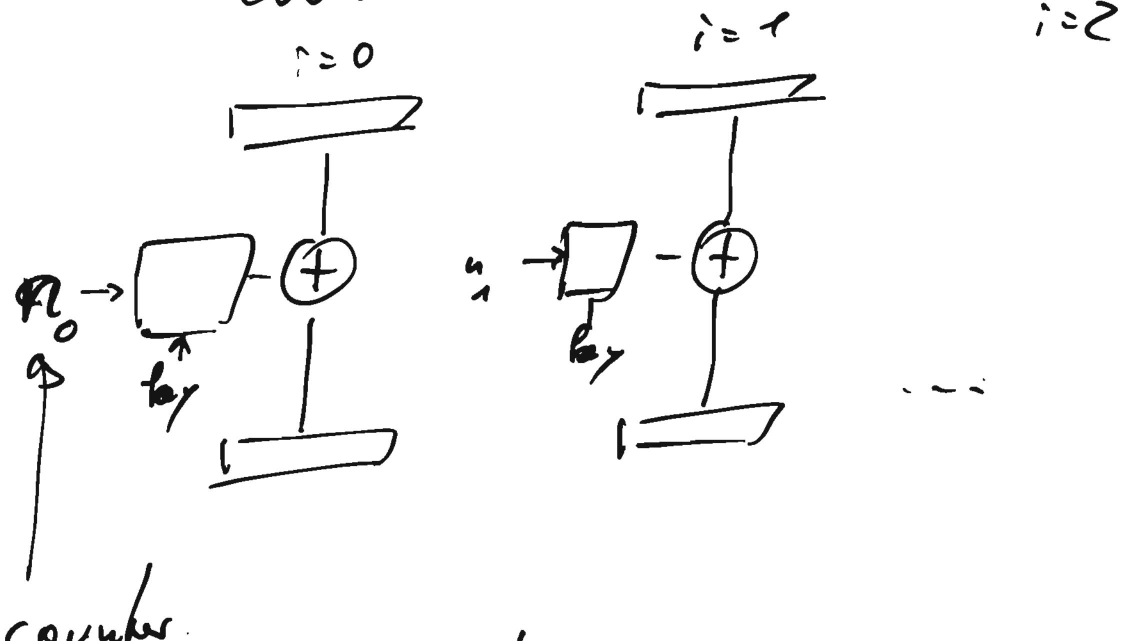
Con: • Problem with asynchronous transfers.

es ec
8.6.10
3

Another solution:

CTR - mode

counter



$n_i = ai + b$

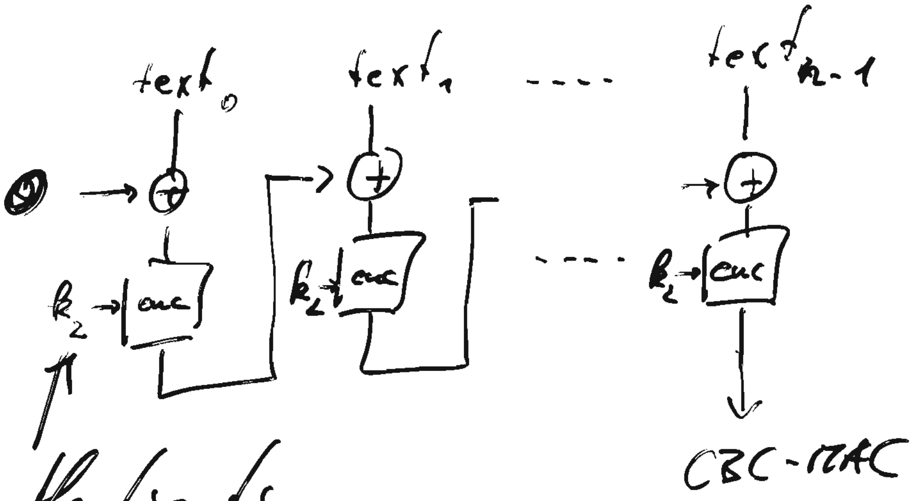
sk. Cke

Pro & Cons : Homework.

Authentication?

Solution 1

CBC-MAC



authentication key!

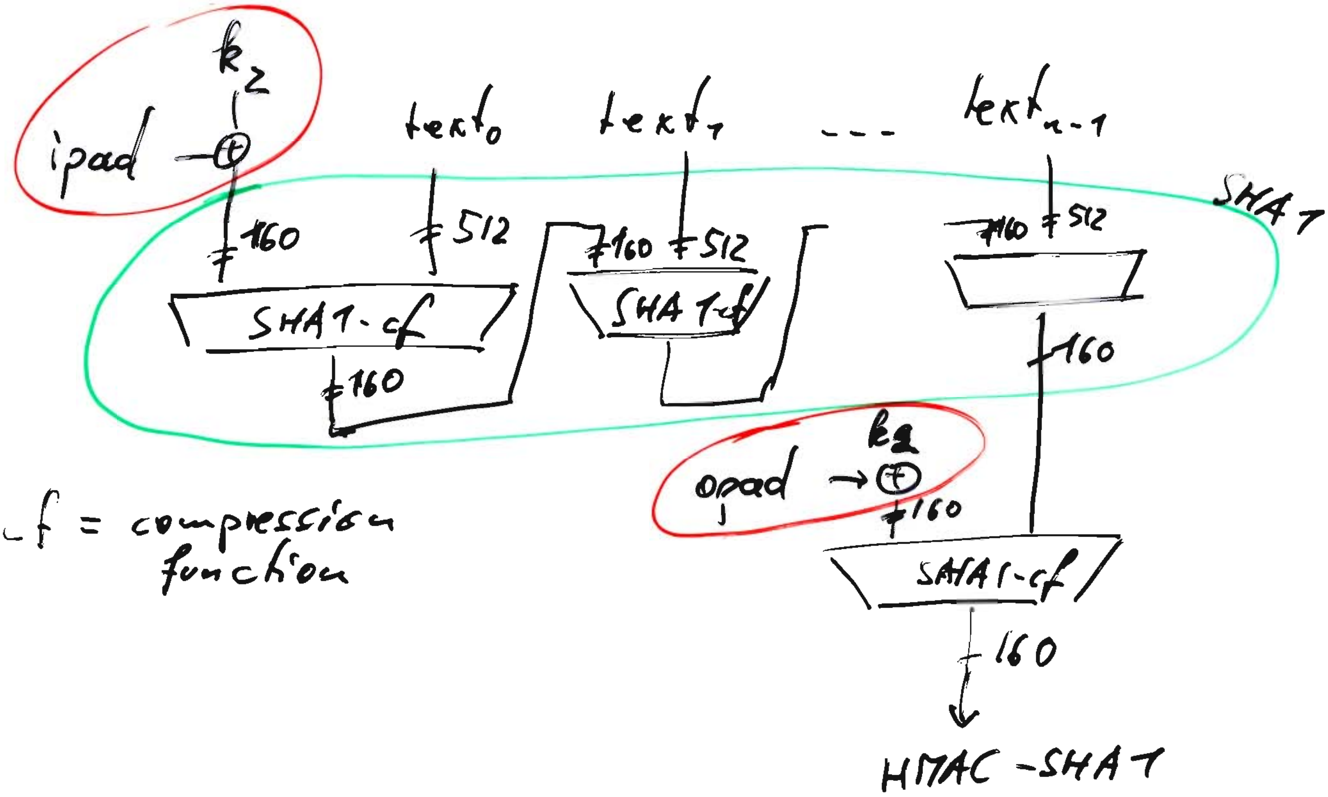
Note:
- an attacker, which is a third party, can neither generate nor check the CBC-MAC value because it depends on a key.
- if any plaintext block is changed then usually the entire CBC-MAC changes.

Need a kind of <u>collision ~~rest~~ resistance</u>.
(Block ciphers do <u>not</u> ~~d~~offer this from their basic definition.)

Solution 2

esec
15.6.10
2

HMAC - SHA1



cf = compression function

Fact One can (almost) prove that this construction is secure if the used hash function is good.

HORTON's principle

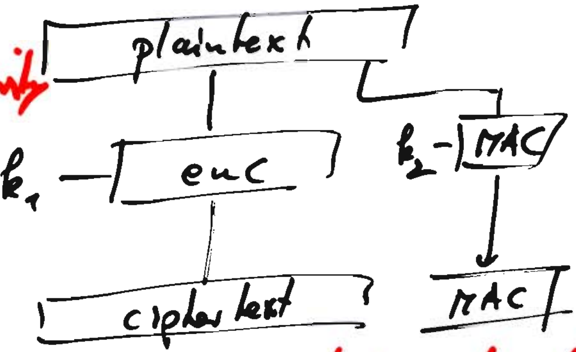
A signature or authentication value must depend on the <u>meaning</u> of the plain text.

→ order of encryption and authentication

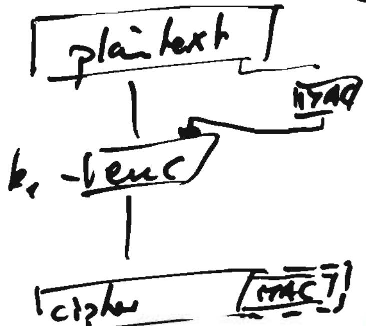
~~AtA~~

AtE „Authenticate then encrypt"

Con: security may be violated even if both, enc and MAC, are good.

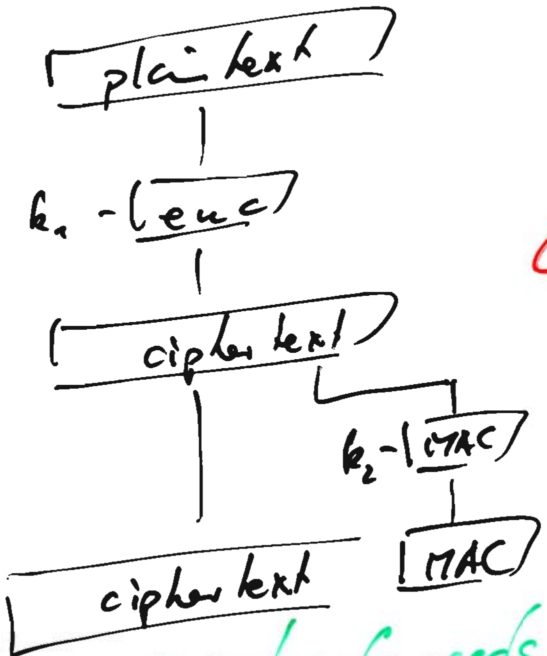


or



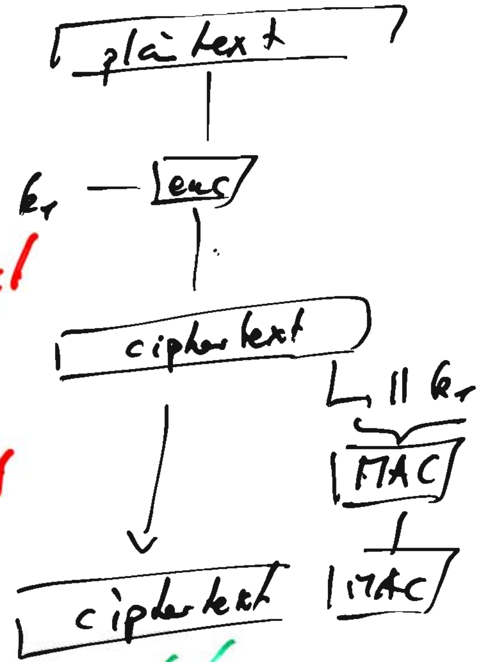
Con: recipient has to decrypt and check the auth. before noticing a bad packet.

EtA „Encrypt then authenticate"



or better

Con: plaintext could be changed by manipulating k_1



Pro: recipient only needs to check authentication to identify bad packet.

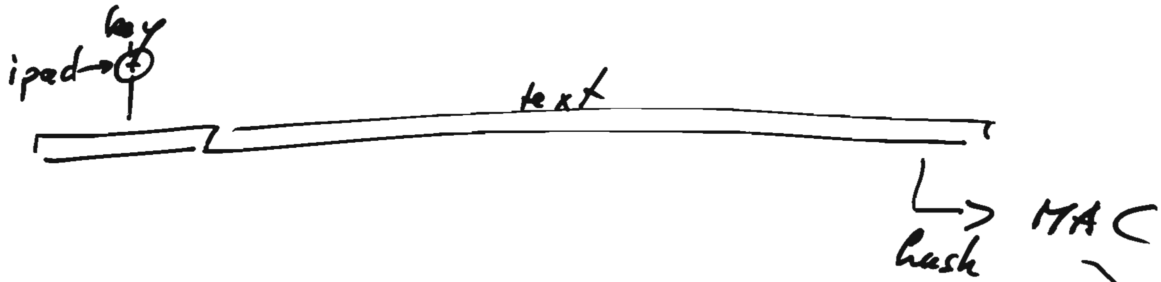
IPsec? EtA as on the left

but k_1 ~~k_2~~ and k_2 are both produced from the same seed (master key) and thus somehow related.

Why do we need a key involved in MAC computation at the least at the beginning and at the end?

1. Can we omit the key at the end?

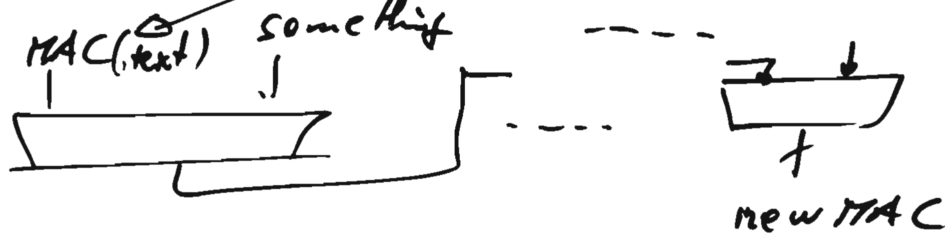
No: EXTENSION Attack:



So to compute the 'MAC' for

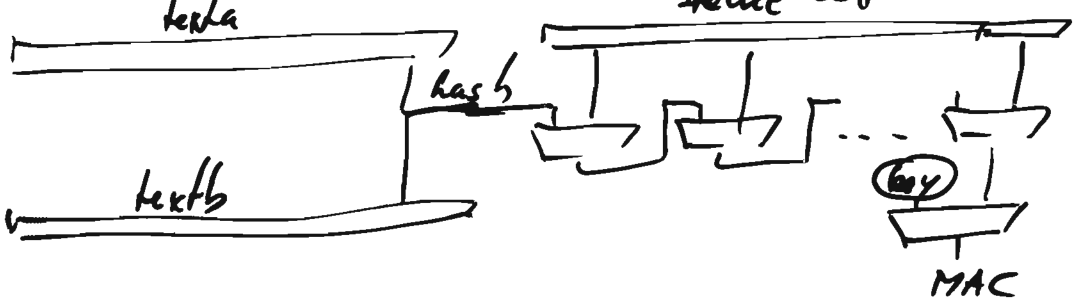
text || something

we just need to compute



2. Can we omit the key at the beginning?

Assume you have a collision for the hash function: texta textb



esec
15.6.10
(5)

Security for (keyed) MAC

An attacker is successful if
he can find a collision
for the MAC without knowing the key

That – due to the ignorance of the key –
is much more difficult than finding
a collision for the hash function.

(3) Why ~~did~~ do ~~we~~ we use the same key
in the beginning and the end?

If half of the bits of the key were
used in the beginning and the other
half at the end

then we could split the attack
versus the two parts of the key
and so get down from

brute force time 2^{160}

down to brute force time $2 \cdot 2^{80}$

for a 160-bit key.

Ex: Look at
AES-XCBC-MAC-96

esec
16.6.10
(1)

Lesson learned (AtE vs. EtA)

It needs extreme care
to combine crypto primitives!

Further keep in mind

- Kerckhoffs' principle
- Horton's principle

esec
16.6.10
2

Elections

~~A democratic election is~~

The aim of an election is

a decision

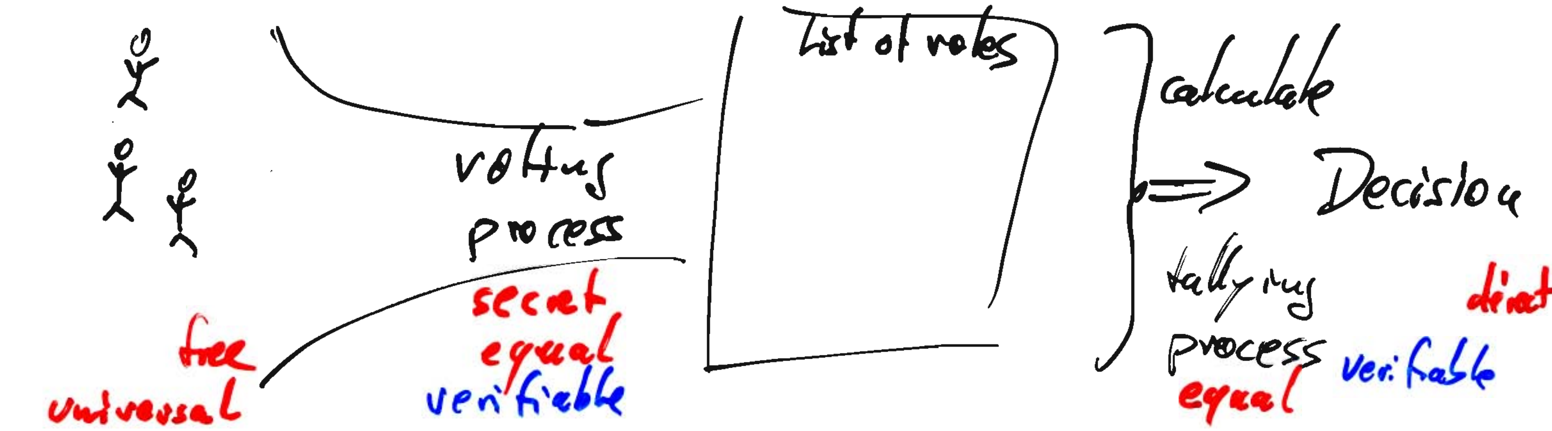
~~by~~ that expresses the opinion of the voters.

We also need that the election is

- free
- fair

The German constitution requires:

- free ✓
- equal ✓
- secret ✓
- universal ✓ (no restriction by race, gender, belief, social status, ...)
- direct ✓



esec
16.6.10
3

Further ~~properties~~ desirable properties:

- publicly verifiable
  - tallying correct
  - certain voter is considered.

Voting process:

- Voter goes to a voting place.
- Officials check whether the voter is the list and allowed to vote. They check the identity of the voter before doing that. Then they mark the voter as "has voted".
- The voter gets the ballot (↑wikipedia), goes to a secret, marks his choice, and puts the closed ballot into the voting booth.

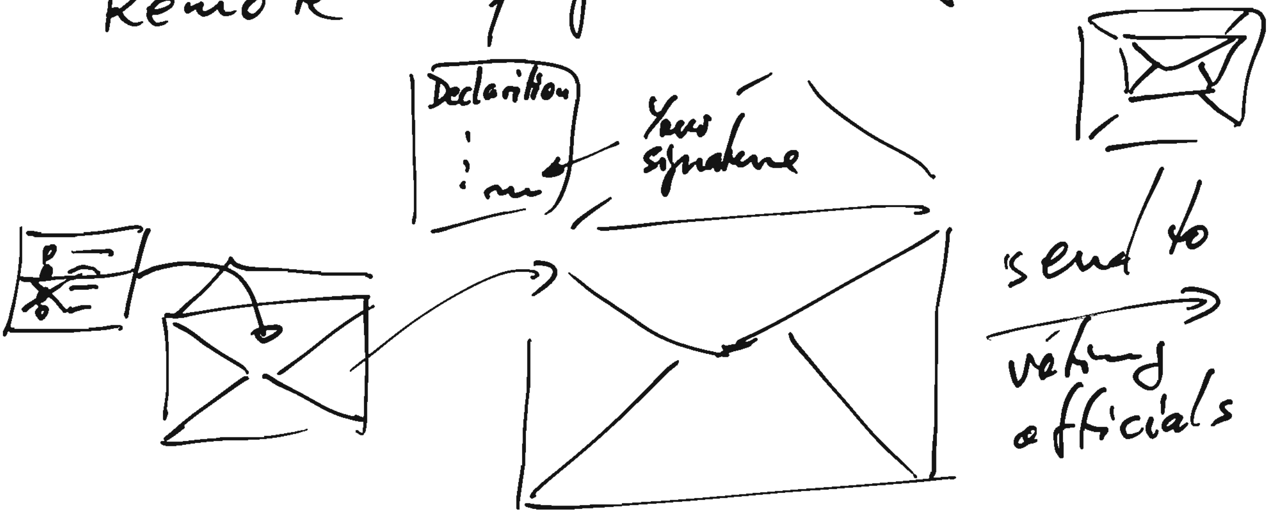
esec
22.6.10
1

Forget voting machines.
We do not talk about them!
We talk about

electronic voting
electronic elections
? remote cryptographic elections ?

~~Classification of schemes:~~

Remote paper voting:



Classification of schemes

- Hidden voter: anonymous-submission of vote
- Hidden vote: encrypted submission of vote
- both.

esec
22.6.10
2

Oldest candidate: Chaum (1981)

Announcement stage

- Chaum's decryption mixnet and its RSA public parameters.
- Each voter is associated with a digital signature.

Registration stage

1. Token generation:
   Each eligible voter V_j generates a random RSA key pair: K_{V_j} public key and ~~token~~ $K_{V_j}^{-1}$ private key.
   Let $token_j \leftarrow K_{V_j}$.
2. The voter V_j sends an encrypted version of his $token_j$ to some official server Mix_1^R together with a signature:

$$E_{K_{\cancel{x}}}(token_j \; \cancel{\#} \; \| \; r_j)$$

random value

and a signature on this.

csec
22.6.10
3

The server Mix_1^R checks the signature and whether it corresponds to an eligible voter that has not voted, yet!

If so it sends a receipt to the voter and it sends a partial decryption

$$D_{K_1^R}\Big(\underbrace{E_K(token_j \,\|\, r_j)}_{= E_{K_1^R}(E_{K_2^R}(\dots E_{K_{\ell-1}^R}(E_{K_\ell^R}(token_j \| r_j^\ell) \| r_j^{\ell-1}) \dots) \| r_j^2) \| r_j^1)}\Big)$$

to Mix_2^R.

Mix_2^R in turn decrypts this and sends

$$D_{K_2^R}\Big(D_{K_1^R}\big(E_K(token_j \,\|\, r_j)\big)\Big)$$

to the next mix.

⋮

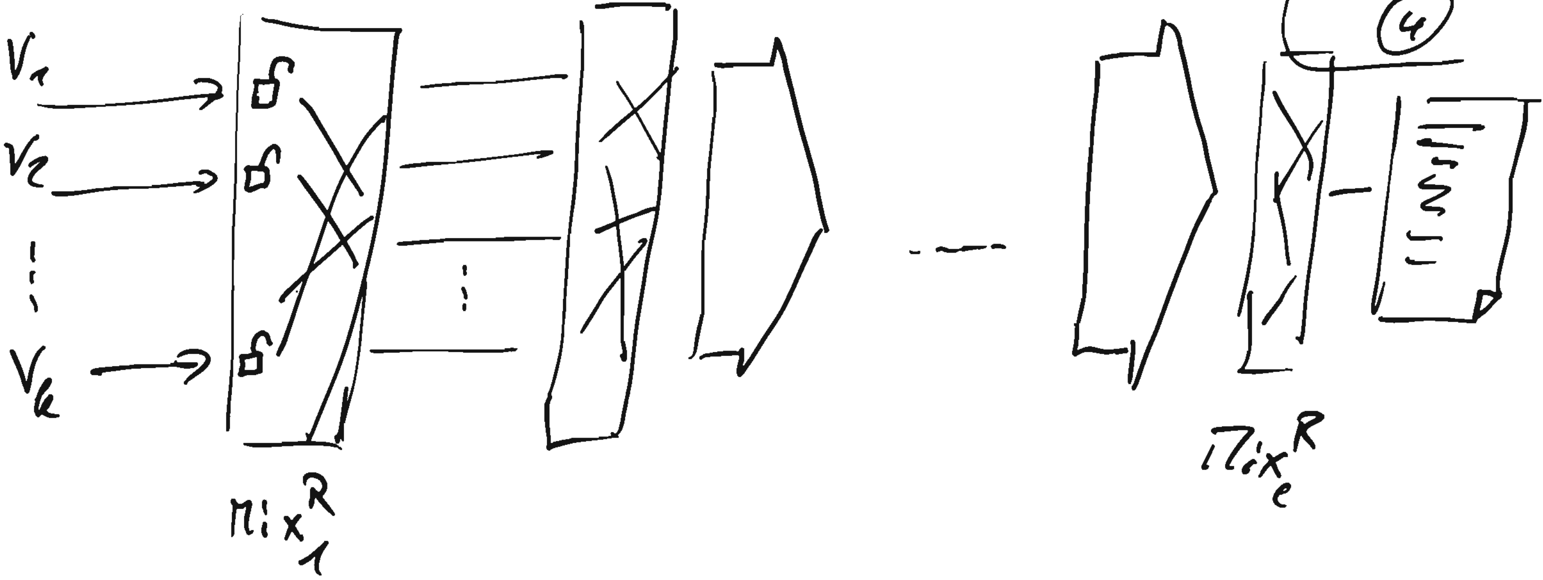
The last mix server obtains

$$token_j$$

and publishes this in a sorted order on 'bulletin board'!

Decryption Mixnet

28.6.10
esec
4



Important: We use a randomised encryption!

Mix_i^R has a key K_i^R, $K_i^{R,-1}$.

and the encryption will add randomness:

$$E_{K_e^R}(\text{token}_j, r_j^e)$$

random bits

$$E_{K_{e-1}^R}(\quad , r_j^{e-1})$$

$\vdots$

$$E_{K_1^R}(\quad , r_j^1)$$

The randomness ensures that an attacker cannot use the decrypted stuff and just encrypt it to see which voter submitted this stuff.

esec
27.6.10
5

For example we could use
RSA + AES:

$$\underbrace{RSA_{K_j}(r), \; AES_r(msg)}_{\overset{!}{=}\; E_{K_j}(msg, r)}$$

Actually, the property that is needed here is INDistinguishability under Adaptive Chosen Ciphertext Attack (IND-CCA2).

Verification stage

The Voter V_j verifies that its $token_j$ arrives on the bulletin board.

We have now achieved that each voter has a key pair whose public key is on the bulletin board.

csec
12.6.10
6

Voting stage

The voter V_j encrypts her vote v_j as

$$\underbrace{E_{K^V_{e-1}}\Big(\underbrace{E_{K^V_e}\big(token_j \,\|\, D_{K_j^{-1}}(v_j \,\|\, 0^k),\ r'_{j,e}\big)}\ ,\ r'_{j,e-1}\Big)}$$

$$\vdots$$

$$E_{K^V_1}\big(\text{----} \ ,\ r'_{j,1} \big)$$

and submits this to the voting decryption mixnet. ~~The first o~~ together with a signature.

The first checks the signature and id of the voter, decrypts one step and sends a bunch to the next mix.

The last mix just puts the now anonymized texts

$$\underbrace{token_j}_{V_j} \,\|\, D_{K_j}(v_j \,\|\, 0^k)$$

esec
22.6.10
7

on another bulletin board list (in sorted order).

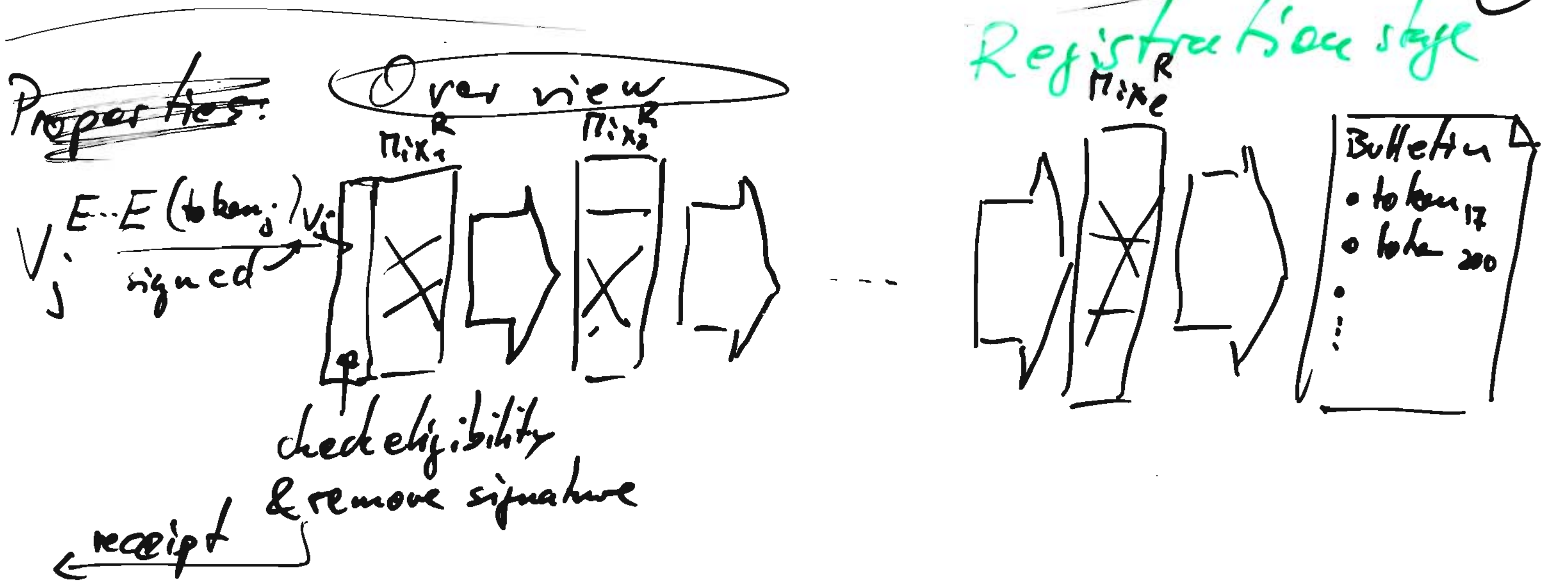
This list now proves that a voter in possession of the secret key corresponding to the $token_j$ which is listed on the first bulletin board has submitted that vote.

Tallying stage

Decrypt and count all votes.

23.6.10
1

Properties: Overview



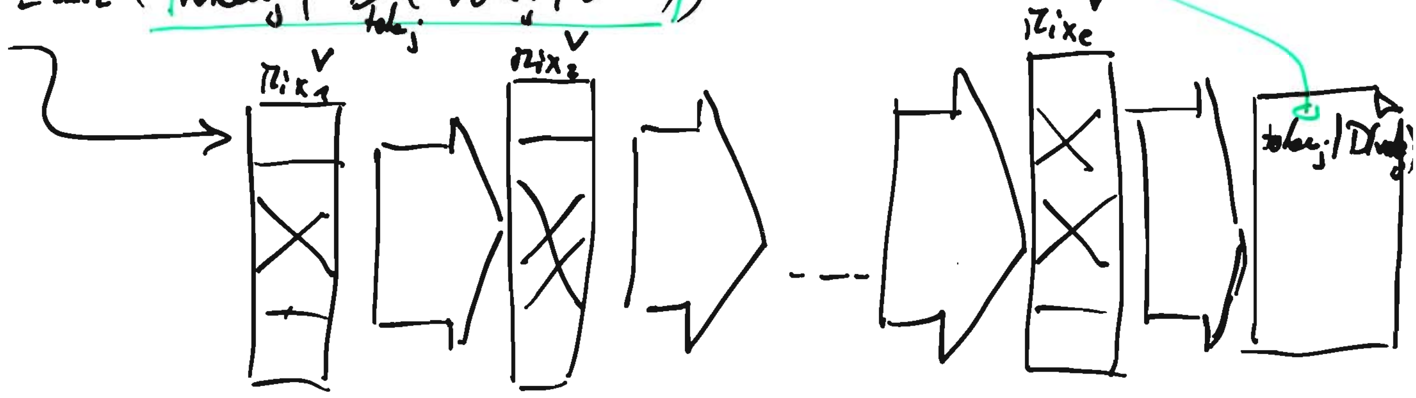
esec
23.6.10
2

~~$V_j \; token_j \mid D(vote_j \mid 0^k)$~~

Voting stage

$E \dots E(\boxed{token_j \mid D_{token_j}(vote_j \mid 0^k)})$

V_j



Properties: general, direct, free, fair and secret.

eligibility check

part of the tallying process and happens <u>after</u> the system has done its work

Yes, provided the secret counterpart of the token remains secret.

BUT:

The key pair of the voter is a receipt for his vote.

The coercer can thus force a voter to any behaviour and afterwards require the key pair as a proof.

esec
23.6.10
3

Eligibility „one man – one vote"

True here.

Anonymity

It is as long as the private key of the voter is not compromised.
But we cannot count on this.
So: NO.

Verifiability

Individual: Yes.

General: Yes, apart from:

a Mix could invalidate votes, though this would be detected by the voter's verification ...

Repair by having every Mix prove that its output corresponds to its input.

Receipt-freeness => Anonymity

Here: NO.

Scalability ✓

Robustness

→ Use a variation of the Mixnet that allows to restrict to, say, five in ten ...
(use secret sharing ...)

1. El Gamal encryption and gimmicks

Algorithm 1.1. El Gamal parameter generation.

Input: Security parameters k, ℓ.
Output: Group G, a prime q, and a generator $P \in G$ of order q.

1. Select a random k-bit prime q.
2. Select an ℓ-bit prime p with $p \equiv_q 1$ and letting $G = \mathbb{Z}_p^\times$ with multiplication. Note that $\#G = p - 1$ and by construction $q \mid p - 1$.
3. Pick a random element P of order q in G. (Pick an arbitrary random element R of G and consider $P = R^{\frac{\#G}{q}}$. If P is the neutral element of G then retry. Otherwise P has order q.)
4. Return (G, q, P).

Note, as of present knowledge, to achieve 80-bit security we need

$$\mathrm{bitlength}(p) = k \approx 1024$$

when choosing $\mathbb{Z}_p$ or a subgroup of $\mathbb{Z}_p$. ElGamal originally proposed to use this with $q = p - 1$. Schnorr and DSA improved this by choosing an element of prime order q, with $\mathrm{bitlength}(q) = \ell \approx 160$. However, all this was before the advent of elliptic curves: with elliptic curves

$$\mathrm{bitlength}(p) = k \approx 160$$

suffices.

Algorithm 1.2. El Gamal parameter generation.

Input: Security parameters k.
Output: Group G, a prime q, and a generator $P \in G$ of order q.

1. Select a random k-bit prime p.
2. Repeat 3–8
3. Select a point $P = (x_P, y_P) \longleftarrow \mathbb{F}_p \times \mathbb{F}_p$.
4. Select a value $a \longleftarrow \mathbb{F}_p^\times$.
5. Set $b = y_P^2 - (x_P^3 + a x_P)$.
6. If $4a^3 + 27b^2 = 0$ in $\mathbb{F}_p$ then try again.
7. Let G be the elliptic curve given by

$$y^2 = x^3 + ax + b$$

over $\mathbb{F}_p$. [Its points are all solutions (x, y) of the equation and a further special point $\mathcal{O}$ at infinity. In particular, P is a point.
Addition of two points Q_1 and Q_2 is essentially defined as follows: consider the line through the points and find the third point Q_3 of intersection with the curve. Define $Q_1 + Q_2 := -Q_3$ by mirroring at the x-axis.]

8. Determine $q = \#G$.
9. Until q prime
10. Return (G, q, P).

Notice that we only need to store x and the "sign" of y to identify a point.

ALGORITHM 1.3. El Gamal key pair generation.

Input: El Gamal parameters (G, q, P).
Output: A key pair with private key $x \in \mathbb{Z}_q$ and public key $X \in G$.

1. Choose $x \longleftarrow \mathbb{Z}_q^{\times}$.
2. Let $X \leftarrow xP$.
3. Return (x,X)

ALGORITHM 1.4. Homomorphic El Gamal encryption.

Publicly known: El Gamal parameters (G, q, P).
Input: The recipient's public key $X \in G$ and the message $M \in G$.
Output: The ciphertext $\mathrm{enc}_X(m)$.

1. Pick a unpredictable temporary private key $t \longleftarrow \mathbb{Z}_q$.
2. Return $(tP, M + tX)$

ALGORITHM 1.5. Homomorphic El Gamal decryption.

Publicly known: El Gamal parameters (G, q, P).
Input: The recipient's private key $x \in \mathbb{Z}_q$, the ciphertext $(T, Y) \in G \times G$.
Output: The plaintext $\mathrm{dec}_x(T, Y)$.

1. Return $Y - xT$

It is easy to check that decrypting returns the original plaintext: Let (T, Y) be a ciphertext of the message M for the recipient with public key X, ie. $T = tP$

and $Y = M + tX$. Note that the public key X is given by the private key x as $X = xP$. Now, the decryption routine returns

$$Y - xT = M + tX - xT = M + txP - xtP = M.$$

Thus the ElGamal scheme works correctly.

Observe that we have

$$\mathrm{dec}_x(m_1 \, \mathrm{enc}_X(M_1) + m_2 \, \mathrm{enc}_X(M_2)) = m_1 M_1 + m_2 M_2.$$

This property is called *homomorphic*: we can combine stuff in the encrypted form and after decription we obtain the corresponding combination of the plaintexts. (In general, it is not necessary that the combination is given by the group operation. Any sort of easily computable combination would do.) As a special case we obtain the reencryption by simply adding an encryption of the neutral element of G, ie. $\mathrm{reenc}_x(M) = \mathrm{enc}_x(M) + \mathrm{enc}_x(\mathcal{O})$.

ALGORITHM 1.6. El Gamal reencryption.

Publicly known: El Gamal parameters (G, q, P).
Input: The recipient's public key $X \in G$ and a ciphertext $(T, Y) \in G \times G$.
Output: A ciphertext $\mathrm{enc}_X(m)$.

1. Pick a unpredictable temporary private key $t' \in \mathbb{Z}_q$.
2. Return $(t'P + T, t'X + Y)$

By the homomorphism property the decryption is $M + \mathcal{O} = M$ again.

2. Non-malleability

The highest security level for encryptions requires that an attacker cannot manipulate messages in a predictable way (non-malleability) under adaptive chosen-ciphertext attacks (NM-CCA2). This is equivalent to the weaker model that an attacker cannot distinguish two self-chosen messages after encryption under adaptive chosen-ciphertext attacks (IND-CCA2). However, it is obvious that the attacker can use the homomorphism property to decrypt without asking the forbidden ciphertext: just add the encryption of a known message M_2, get the decryption from the oracle, and finally subtract M_2. Thus the attacker gets the decryption, can thus easily determine which of his two self-chosen messages was encrypted, and thus wins the game.

To spoil this attack various proposals have been made. One consists in signing the ciphertext with a Schnorr signature:

Algorithm 2.1. Non-malleable El Gamal encryption.

Publicly known: El Gamal parameters (G, q, P).
Input: The recipient's public key $X \in G$, the message $M \in G$.
Output: The ciphertext $\mathrm{nmenc}_X(m)$.

1. Pick two random temporary keys $t, u \longleftarrow \mathbb{Z}_q$.
2. Encrypt $(T, Y) \leftarrow (tP, M + tX)$.
3. Compute a challenge $c \leftarrow \mathbb{Z}_q(\mathrm{hash}(uP, T, Y)) \in \mathbb{Z}_q$.
4. Compute the response $r \leftarrow u + ct$ in Z_q.
5. Return (T, Y, c, r)

Algorithm 2.2. Non-malleable El Gamal decryption.

Publicly known: El Gamal parameters (G, q, P).
Input: The recipient's private key $x \in \mathbb{Z}_q$, the ciphertext $(T, Y, c, r) \in G \times G \times \mathbb{Z}_q \times \mathbb{Z}_q$.
Output: The plaintext $\mathrm{nmdec}_x(T, Y, c, r)$.

1. Compute $U \leftarrow rP - cT$ and $c' \leftarrow \mathbb{Z}_q(\mathrm{hash}(U, T, Y)) \in \mathbb{Z}_q$.
2. If $c' \neq c$ then Return Failure
3. Return $Y - xT$

Notice that Algorithm 2.1 step 3–4 and the verification $c' \stackrel{?}{=} c$ in Algorithm 2.2 form a non-interactive proof of knowledge for the discrete logarithm t of T with respect to P.

Actually, the attacker's task would be to — say — reencrypt (T, Y, c, r). He can of course easily present (T', Y') with the same plain text. However, constructing (c', r') as well would be a proof of knowledge of the discrete logarithm of T' with respect to P, and thus (as the attacker chooses $T' - T$) of the discrete logarithm of T with respect to P. So either the attacker is the sender or he can break the DLP. But we assume he cannot. This reasoning however neglects possible effects of the choice of c as the value of a hash function.

3. A zero-knowledge argument

PROTOCOL 3.1. Interactive zero-knowledge proof of equalitiy of discrete logarithms.

Publicly known: El Gamal parameters (G, q, P).
Public input: Group elements $P, T, X, Y \in G$.
Private input to the prover: The discrete logarithm t of T wrt. P and of Y wrt. X, ie. $t \in \mathbb{Z}_q$ such that $T = tP$ and $Y = tX$.

1. The prover chooses a temporary private key $u \longleftarrow \mathbb{Z}_q$ and computes $U \leftarrow uP$ and $V \leftarrow uX$ in G. She sends U and V to the verifier. $\xrightarrow{\quad (U,V) \quad}$
2. The verifier chooses a challenge $c \longleftarrow \mathbb{Z}_q$ and sends it to the prover. $\xleftarrow{\quad c \quad}$
3. The prover computes the response $r \leftarrow u + ct$ and sends it to the verifier. $\xrightarrow{\quad r \quad}$
4. The verifier checks that $rP = U + cT$ and $rX = V + cY$.

An interactive zero-knowledge proof is a protocol with the properties

(computational) completeness If both parties, Paula and Victor, are honest the verifier (almost) always accepts.

(computational) soundness If the prover Patrick cheats the (honest) verifier Victor almost never accepts.

(computational) zero-knowledge Even if the verifier Vlad cheats he can still not learn anything. That is, whatever the verifier Vlad can compute after a conversation he can also compute without a conversation.

This is usually established by the existence of a simulator which produces a transcript that looks like a conversation and the probabilities for the transcripts are (almost) the same as the probabilities for the conversations.

Actually, in the following we restrict mainly to the case of a *semi-honest* verifier: Vlad is allowed to learn from the protocol but otherwise follows exactly the honest verifier Victor's algorithm. The semi-honest Vlad definitely does not choose c depending on (U, V).

We assume that all parties are randomized polynomial time bounded. Each computation may fail with negligible probability.

esec
30.6.10
7

completeness

If the prover's claim holds
(i.e. the prover is honest)
and the verifier follows his algorithms
(i.e. the verifier is honest)
then the verifier always accepts.

Proof: $U + cT = uP + ctP$
$= (u+ct)P = rP.$

$V + cY = uX + ctX$
$= (u+ct)X = rX.$ ✓ □

soundness

Assume the prover cheats
and $T = tP$, $Y = t'X \neq tX$.

Next, the prover has to choose U and V,
say $U = uP$, $V = u'X$.

Only now — after U, V are fixed — the verifier sends his challenge c.

Now $U + cT = (u+ct)P$
and so the prover has to $r = u + ct$ to satisfy
the verifier's first check.

But
$V + cY = (u' + ct')X$

esec
30.6.10
2

The verifier's second check is true
iff $u' + c t' = r$

Or: both checks are true iff

$$r = u + ct$$
$$r = u' + ct'$$

If the prover can satisfy this for only two different values of c
then $u = u'$ and $t = t'$.

[
$r_1 = u_1 + c_1 t$ $\qquad r_2 = u + c_2 t$
$r_1 = u' + c_1 t'$ $\qquad r_2 = u' + c_2 t'$

(c_1, r_1) $\qquad (c_2, r_2)$
These are two points on both lines and thus the lines must coincide, i.e. $u = u'$, $t = t'$.]

Thus
prob(cheating Paula convinces Victor)

$$= \frac{1}{q} \; (\approx 2^{-160}),$$

this is exponentially small. □

esec
30.6.10
3

(computational) honest-verifier zero-knowledge:

whatever the verifier V could compute
after a conversation <P, V>

he could also compute
after a simulation SAM

where

SAM
Inputs: P, T, X, Y (public input and private input the verifier)

Output: A protocol ~~of something~~ that
Looks like a protocol of a
conversation.

1.

2. $c \xleftarrow{R} \mathbb{Z}_q$.

3. $r \xleftarrow{R} \mathbb{Z}_q$.

4. $U \leftarrow rP - cT$,
$V \leftarrow rX - cY$.

5.

6. Return $((U, V), c, r)$

Notice that SAM's output would always
pass the honest verifier VICTOR's checks:

$$rP = U + cT,$$
$$rX = V + cY.$$

Actually, we have to consider the distribution of possible outputs:

esec
30.6.10
4

$$\text{prob}\left(\langle P, V \rangle = \left((U_0, V_0), c_0, r_0 \right) \right) = \frac{1}{q^2}$$

$$\text{prob}\left(S^M = \left((U_0, V_0), c_0, r_0 \right) \right) = \frac{1}{q^2}$$

for a given (U_0, V_0), c_0, r_0 fulfilling the verifier's checks, ie.

$$r_0 P = U_0 + c_0 T,$$
$$r_0 X = V_0 + c_0 Y.$$

$\square$

General zero-knowledge:

Problem a dishonest verifier V' may choose c non-uniform or worse – depending on U, V.

As long as V' chooses c independent of U, V we can just replace the choice in the simulator. But if the choice depends on U, V then ~~things~~ become tricky.

We make it non-interactive by the Fiat & Shamir (1986) heuristic: replace the random challenge sent by the verifier with a deterministic computation whose outcome is unpredictable to the prover even if she messes around with the entire variables at her disposal. (Actually, one can always transform a proof of knowledge into one where the verifier only sends random bits. But we have that already.) We obtain:

Protocol 3.2. Non-interactive zero-knowledge proof of equalitiy of discrete logarithms.

Publicly known: El Gamal parameters (G, q, P).
Public input: Group elements $P, T, X, Y \in G$.
Private input to the prover: The discrete logarithm t of T wrt. P and of V wrt. U, ie. $t \in \mathbb{Z}_q$ such that $T = tP$ and $Y = tX$.

1. The prover chooses a temporary private key $u \longleftarrow \mathbb{Z}_q$ and computes $U \leftarrow uP$ and $V \leftarrow uX$ in G. She sends U and X to the verifier. $\xrightarrow{\qquad (U, V) \qquad}$
2. The prover computes a challenge $c \leftarrow \mathbb{Z}_q(\mathrm{hash}(T, Y, U, V))$ and sends it to the verifier. $\xrightarrow{\qquad c \qquad}$
3. The prover computes the response $r \leftarrow u + ct$ and sends it to the verifier. $\xrightarrow{\qquad r \qquad}$
4. The verifier checks that $rP = U + cT$, $rX = V + cY$ and $c = \mathbb{Z}_q(\mathrm{hash}(T, Y, U, V))$.

We can further simplify this by dropping (U, V) from the messages since they can be reconstructed from c and r easily, a computation that the verifier must perform anyways. Thus in the last step the verifier only checks

$$c = \mathbb{Z}_q(\mathrm{hash}(T, Y, rP - cT, rX - cY)).$$

4. A proof of knowledge

PROTOCOL 4.1. Interactive proof of knowledge of a discrete logarithm.

Publicly known: El Gamal parameters (G, q, P).
Public input: Group elements $P, T \in G$.
Private input to the prover: The discrete logarithm of T wrt. P, ie. $t \in \mathbb{Z}_q$ such that $T = tP$.

1. The prover chooses a temporary private key $u \longleftarrow \mathbb{Z}_q$ and computes $U \leftarrow uP$ in G. She sends U to the verifier. $\xrightarrow{\qquad U \qquad}$
2. The verifier chooses a challenge $c \longleftarrow \mathbb{Z}_q$ and sends it to the prover. $\xleftarrow{\qquad c \qquad}$
3. The prover computes the response $r \leftarrow u + ct$ and sends it to the verifier. $\xrightarrow{\qquad r \qquad}$
4. The verifier checks that $rP = U + cT$.

A proof of knowledge is an interactive zero-knowledge protocol with the additional property

proof of knowledge A cheating verifier that can talk to the same(!) prover several times can extract the knowledge from the conversations. Here, same prover means that the prover is using the same random bits again.

Protocol 4.1 is a proof of knowledge.

Pf Define the knowledge extractor as follows:

Paula		KnightEgon
	$\xleftarrow{\text{reset}}$	
$u \in_R \mathbb{Z}_q$ $U = uP$	$\xrightarrow{U}$	
		$c_1, c_2 \in_R \mathbb{Z}_q$ $c_1 \neq c_2$
	$\xleftarrow{c_1}$	
$r_1 \leftarrow u + c_1 t$	$\xrightarrow{r_1}$	
	$\xleftarrow{\text{reset}}$	
same u, same U.	$\xrightarrow{U}$	
	$\xleftarrow{c_2}$	
$r_2 \leftarrow u + c_2 t$	$\xrightarrow{r_2}$	

$$r_1 P \stackrel{?}{=} U + c_1 T$$
$$r_2 P \stackrel{?}{=} U + c_2 T$$

Thus we know:

$$r_1 = u + c_1 t \quad \text{in } \mathbb{Z}_q$$
$$r_2 = u + c_2 t \quad \text{in } \mathbb{Z}_q$$

and

$$r_2 - r_1 = \underbrace{(c_2 - c_1)}_{\neq 0} t \quad \text{in } \mathbb{Z}_q$$

and

$$\frac{r_2 - r_1}{c_2 - c_1} = t \quad \text{in } \mathbb{Z}_q$$

Output t. □

Again, we make it non-interactive by the Fiat & Shamir (1986) heuristic. We obtain:

Protocol 4.2. Non-interactive proof of knowledge of a discrete logarithm.

Publicly known: El Gamal parameters (G, q, P).
Public input: Group elements $P, T \in G$.
Private input to the prover: The discrete logarithm of T wrt. P, ie. $t \in \mathbb{Z}_q$ such that $T = tP$.

1. The prover chooses a temporary private key $u \longleftarrow \mathbb{Z}_q$ and computes $U \leftarrow uP$ in G. She sends U to the verifier. $\xrightarrow{\quad U \quad}$
2. The prover computes a challenge $c \leftarrow \mathbb{Z}_q(\mathrm{hash}(T, U))$ and sends it to the verifier. $\xrightarrow{\quad c \quad}$
3. The prover computes the response $r \leftarrow u + ct$ and sends it to the verifier. $\xrightarrow{\quad r \quad}$
4. The verifier checks that $rP = U + cT$ and $c = \mathbb{Z}_q(\mathrm{hash}(T, U))$.

Clearly, the prover can send everything together in a single message (U, c, r).

As earlier we can drop U from the messages and instead recompute it and check

$$c = \mathbb{Z}_q(\mathrm{hash}(T, rP - cT)).$$

We could instead also drop c and reconstruct that, but for many groups you need more bits to store U than you need to store c.

5. Distributed keys

PROTOCOL 5.1. Distributed key generation.

Publicly known: El Gamal parameters (G, q, P).
Input to S_i: Id i and connections to all other share holders S_j.
Private output to S_i: Private key shares x_i.
Output: A public key X, and public key shares X_i.

1. Share holder S_i chooses a private key share $x_i \xleftarrow{} \mathbb{Z}_q$ and compute $X_i \leftarrow x_i P \in G$.
2. Share holder S_i publishes (ie. sends to all other share holders) a commitment $\mathrm{hash}(X_i)$ on its public key share X_i.
3. Wait until all share holders are done so far.
4. Share holder S_i publishes X_i and proves knowledge of x_i non-interactively, ie. publishes $\mathrm{KnowDlog}(P, X_i)$.
5. Wait until all share holders are done so far.
6. Each share holder checks all commitments and proofs. If something cannot be verified, shout and stop.
7. Return $X = \sum_i X_i$, $(X_i)_i$

The sender merely encrypts his message with the shared public key X. However, as long as one share holder is honest, the corresponding private key $x = \sum_i x_i$ is not known to any entity. To decrypt all share holders have to work together again:

PROTOCOL 5.2. Distributed decryption.

Publicly known: El Gamal parameters (G, q, P).
Input: The ciphertext $(T, Y) \in G \times G$, and the public shares X_i.
Private inputs: Share holder S_i gets its private key share x_i.
Output: $\mathrm{DistDec}_{(x_i)_i}(T, Y)$.

1. Share holder S_i computes and publishes $T_i \leftarrow x_i T$ and proves equality of discrete logarithms of T_i wrt. X_i and T wrt. P, ie. $\mathrm{EqDlog}(P, T, X_i, T_i)$. $\xrightarrow{T_i,\ \mathrm{EqDlog}(\dots)}$
2. Wait until all share holders are done so far. $\xleftarrow{\qquad\qquad}$
3. Each share holder checks all proofs. If something cannot be verified, shout and stop.
4. Compute $M \leftarrow Y - \sum T_i$.
5. Return M

Important: in both protocols no share holder learns private key shares of other (honest) share holders. [Proof: Exercise.]

6. A more sophisticated zero-knowledge proof

The problem in remote elections is that nobody can see whether the voter is under pressure during his voting. So the above zero-knowledge proof is actually too good, as also a coercer will be convinced by such a proof if he is standing "behind" the voter. But we can do better: The following two zero-knowledge proofs prove the statement:

> The El Gamal ciphertexts (T, Y) and (T', Y') encrypt the same message (for the recipient with public key X)
>
> *or*
>
> the prover knows the voter's private key.

This statement can be proved by the party that generated (T, Y) from (T', Y') or it can be proved by the voter. As zero-knowledge proofs are always witness-indistinguishable, a coercer in the role of the verifier cannot tell which of the two forms he sees.

PROTOCOL 6.1. Interactive designated verifier proof.

Publicly known: El Gamal parameters (G, q, P).

Public input: Group elements $T, Y, T', Y' \in G$ and the public key X_{vid} of the voter vid.

Private input to the prover: The reencryption randomness $z \in \mathbb{Z}_q$ such that $T' - T = zP$ and $Y' - Y = zX$.

1. The prover chooses temporary private keys $s, t, w \xleftarrow{} \mathbb{Z}_q$ and computes in G
   - $\widetilde{T} \leftarrow sP$,
   - $\widetilde{Y} \leftarrow sX$ and
   - $\widetilde{V} \leftarrow tP + wX_{\text{vid}}$.

   She sends $\widetilde{T}$, $\widetilde{Y}$ and $\widetilde{V}$ to the verifier. $\xrightarrow{(\widetilde{T}, \widetilde{Y}, \widetilde{V})}$
2. The verifier chooses a challenge $c \xleftarrow{} \mathbb{Z}_q$ and sends it to the prover. $\xleftarrow{\quad c \quad}$
3. The prover computes the response $r \leftarrow s + z(c + w)$ and sends it to the verifier. $\xrightarrow{(r, t, w)}$

4. The verifier computes
   - $\widetilde{T}' \leftarrow rP - (c+t)(T'-T)$,
   - $\widetilde{Y}' \leftarrow rX - (c+t)(Y'-Y)$ and
   - $\widetilde{V}' \leftarrow tP + wX_{\text{vid}}$.

   He checks whether $\widetilde{T}' \stackrel{?}{=} \widetilde{T}$, $\widetilde{Y}' \stackrel{?}{=} \widetilde{Y}$, and $\widetilde{V}' \stackrel{?}{=} \widetilde{V}$.

PROTOCOL 6.2. Interactive fake designated verifier proof.

Publicly known: El Gamal parameters (G, q, P).
Public input: Group elements $T, Y, T', Y' \in G$ and the public key X_{vid} of the voter vid.
Private input to the prover: The verifier's private key x_{vid}.

1. The prover chooses the response $r \longleftarrow \mathbb{Z}_q$ and random values $a, v \longleftarrow \mathbb{Z}_q$ and computes in G
   - $\widetilde{T} \leftarrow rP - a(T'-T)$,
   - $\widetilde{Y} \leftarrow rX - a(Y'-Y)$ and
   - $\widetilde{V} \leftarrow vP$.

   She sends $\widetilde{T}$, $\widetilde{Y}$ and $\widetilde{V}$ to the verifier. $\xrightarrow{(\widetilde{T}, \widetilde{Y}, \widetilde{V})}$
2. The verifier chooses a challenge $c \longleftarrow \mathbb{Z}_q$ and sends it to the prover. $\xleftarrow{c}$
3. The prover computes $t \leftarrow a - c$, $w \leftarrow (v-t)x_{\text{vid}}^{-1}$ in $\mathbb{Z}_q$ and sends (r, t, w) to the verifier. $\xrightarrow{(r,t,w)}$
4. The verifier computes
   - $\widetilde{T}' \leftarrow rP - (c+t)(T'-T)$,
   - $\widetilde{Y}' \leftarrow rX - (c+t)(Y'-Y)$ and
   - $\widetilde{V}' \leftarrow tP + wX_{\text{vid}}$.

   He checks whether $\widetilde{T}' \stackrel{?}{=} \widetilde{T}$, $\widetilde{Y}' \stackrel{?}{=} \widetilde{Y}$, and $\widetilde{V}' \stackrel{?}{=} \widetilde{V}$.

By the Fiat & Shamir (1986) heuristic we can again transform both into a non-interactive protocol:

PROTOCOL 6.3. Non-interactive designated verifier proof.

Publicly known: El Gamal parameters (G, q, P).
Public input: Group elements $T, Y, T', Y' \in G$ and the public key X_{vid} of the voter vid.
Private input to the prover: The reencryption randomness $z \in \mathbb{Z}_q$ such that $T' - T = zP$ and $Y' - Y = zX$.

1. The prover chooses temporary private keys $s, t, w \longleftarrow \mathbb{Z}_q$ and computes in G
   - $\widetilde{T} \leftarrow sP$,
   - $\widetilde{Y} \leftarrow sX$ and
   - $\widetilde{V} \leftarrow tP + wX_{\text{vid}}$.

   She sends $\widetilde{T}$, $\widetilde{Y}$ and $\widetilde{V}$ to the verifier.
2. The prover computes a challenge

   $$c \leftarrow \mathbb{Z}_q(\text{hash}(T, Y, T', Y', \widetilde{T}, \widetilde{Y}, \widetilde{V}))$$

   and sends it to the verifier. $\xrightarrow{\quad c \quad}$
3. The prover computes the response $r \leftarrow s + z(c + w)$ and sends it to the verifier. $\xrightarrow{\quad (r, t, w) \quad}$
4. The verifier computes
   - $\widetilde{T}' \leftarrow rP - (c + t)(T' - T)$,
   - $\widetilde{Y}' \leftarrow rX - (c + t)(Y' - Y)$ and
   - $\widetilde{V}' \leftarrow tP + wX_{\text{vid}}$.

   He checks whether $\widetilde{T}' = \widetilde{T}$, $\widetilde{Y}' = \widetilde{Y}$, and $\widetilde{V}' = \widetilde{V}$ by computing

   $$c' \leftarrow \mathbb{Z}_q(\text{hash}(T, Y, T', Y', \widetilde{T}', \widetilde{Y}', \widetilde{V}'))$$

   and checking $c' \stackrel{?}{=} c$.

Protocol 6.4. Non-interactive fake designated verifier proof.

Publicly known: El Gamal parameters (G, q, P).
Public input: Group elements $T, Y, T', Y' \in G$ and the public key X_{vid} of the voter vid.
Private input to the prover: The verifier's private key x_{vid}.

1. The prover chooses the response $r \longleftarrow \mathbb{Z}_q$ and random values $a, v \longleftarrow \mathbb{Z}_q$ and computes in G
   - $\widetilde{T} \leftarrow rP - a(T' - T)$,
   - $\widetilde{Y} \leftarrow rX - a(Y' - Y)$ and
   - $\widetilde{V} \leftarrow vP$.

   She sends $\widetilde{T}$, $\widetilde{Y}$ and $\widetilde{V}$ to the verifier.
2. The prover computes a challenge

$$c \leftarrow \mathbb{Z}_q(\mathrm{hash}(T, Y, T', Y', \widetilde{T}, \widetilde{Y}, \widetilde{V}))$$

   and sends it to the verifier. $\xrightarrow{\quad c \quad}$
3. The prover computes $t \leftarrow a - c$, $w \leftarrow (v - t)x_{\mathrm{vid}}^{-1}$ in $\mathbb{Z}_q$ and sends (r, t, w) to the verifier. $\xrightarrow{(r,t,w)}$
4. The verifier computes
   - $\widetilde{T}' \leftarrow rP - (c + t)(T' - T)$,
   - $\widetilde{Y}' \leftarrow rX - (c + t)(Y' - Y)$ and
   - $\widetilde{V}' \leftarrow tP + wX_{\mathrm{vid}}$.

   He checks whether $\widetilde{T}' = \widetilde{T}$, $\widetilde{Y}' = \widetilde{Y}$, and $\widetilde{V}' = \widetilde{V}$ by computing

$$c' \leftarrow \mathbb{Z}_q(\mathrm{hash}(T, Y, T', Y', \widetilde{T}', \widetilde{Y}', \widetilde{V}'))$$

   and checking $c' \stackrel{?}{=} c$.

Civitas

esec
13.7.10
①

Security model

- Need compromise!

Properties:

(1.) VERIFYABILITY

→ Voter verifiability

→ Universal verifiability

Need a formal definition..

(2.) COERCION RESISTANCE:
A Voter cannot prove whether or how they voted <u>even</u> if they interact with the attacker during voting.

\+ AVAILABILITY

\+ SCALABILITY

esec
13.7.10
2

Threat model, attacker properties

- The attacker can corrupt some but not all "authorities" / non-voter-components. We assume a threshold. (Like: he can corrupt 50% of them ...)

- The attacker may coerce voters, demand their secrets, demand any behaviour of them — remotely or like physical presence of the voters. But the adversary may not control a voter throughout an entire election. (Otherwise the voter could never register or vote.)

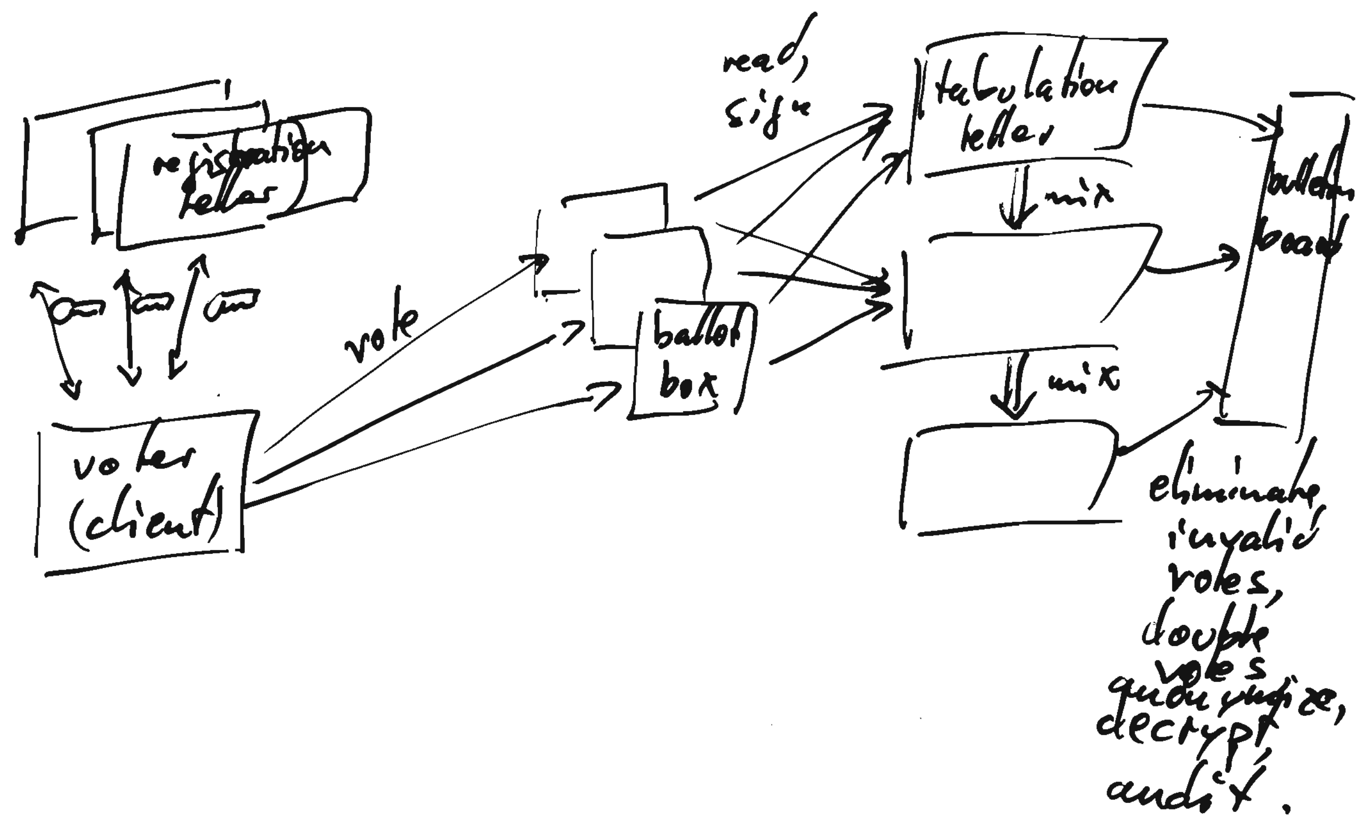
- The adversary may control all public channels on the network.
  However, we assume the existence of some anonymous channel on which the adversary cannot identify the sender.
  And some untappable channel which the adversary cannot use at all.

- The adversary may perform any polynomial-time computation.

Design

Agents:

- voter
- supervisor
  administers the election including ballot design, specifying the tellers, and starting and stopping.
- registrar.
  authorizes votes
- registration tellers that generate credentials that voters use to cast their votes
- tabulation tellers tally votes.



csec
14.7.10
(4)

There are several bulletin boards, ie. insert-only, readable memory.

Namely:
- the ballot boxes
- the broadcast bulletin board.

Setup phase

1. The supervisor creates the election:
   - post ballot design.
   - identifies the tellers by posting their individual public keys. (and ballot boxes)
   - post the electoral role, ie. the list of all voters together with their registration and designation keys.
2. The tabulation tellers generate a distributed key pair X_{TT}, (x_{TT_i}), and post the public key.
3. The registration tellers RT_i generate private credentials $s_{i,vid}$ for each voter vid, and publish an encryption of $s_{i,vid}$.

esec
14.7.10
2

Registration & voting phase

Registration means that the voter talks to all his registration tellers and acquire his private credentials: $(s_{i,vid})_i$ and stores his private credential $s_{vid} = \prod_i s_{i,vid}$.

Voting means that the voter sends

$$\Big(\mathrm{Enc}(s; X_{TT}),$$
$$\mathrm{Enc}(v; X_{TT}),$$

proof that v is one of the specified voting choices, (zk!)
proof that the voter 'knows' s and v.)

to some ballot boxes of his choice.

How to resist coercion?

If the adversary demands that the voter	Then the voter
... submits a particular vote	... does it with fake credentials.
... sell or surrender a credential	... supplies fake credentials.
... abstains (and give him credentials to check...)	... supplies fake credentials and votes.

esec
14.7.10
3

Revoting?

A revoting policy may allow several revotes.
However, this requires some form of ordering information to be able to decide which vote is the first or last or... one, resp..

Timing information on the ballot boxes might be used.
But: you may want the voter to proof in a revote that he knows the content of earlier votes and indicate which these are.

→ Room for discussion.

Ballot design and tallying definition:

- plurality vote : 1 out of N.
- approval vote : any subset of N.
- ranked vote : order the N options.
- write-in votes.

Danger:
write-in votes and also approval and ranked votes allow for a covert channel!

Look up ARROWS theorem.

Tabulation phase

es ec
14.7.10
4

1. Retrieve data.
   TT_i reads all votes from all ballot boxes.
   and reads the public credentials.

2. Verify proofs.
   TT_i check each vote to verify well-
   formedness. Any vote with an
   invalid proof is discarded.

3. Eliminate duplicates

   The TTs run "plain text equivalence
   tests" for any pair of votes on the
   credential encryption. Votes with dup-
   licate ~~cr~~ credentials are elimanited
   according to the revoting policy.

   voting
   $O((\#\text{cast votes})^2)$

4. Anonymize.
   Mixing with proofs of
   correct mixing.

5. Eliminate fake credential votes.

   $O(\#\text{cast votes} \cdot \#\text{voters})$

6. Decrypt.

esec
20.7.10
1

Security evaluation

Trust assumption 1

The adversary cannot simulate a voter during registration.

→ Defense:
- long registration period.
- at least one physical registration teller.
- tamper resistant hardware
  (↑ Estonia id card)

Trust assumption 2

Each voter trusts at least one registration teller, and the channel to the voter's trusted registration teller is untappable.

→ Defense:
- physical registration tellers most likely offer this

Trust assumption 3

Voters trust their voting clients.

→ Defense:
- use additional hardware
  (like temper resistant smart card readers with display ...)

- open source
- "own" source

} code.

OPEN RESEARCH!

eser
20.7.10
2

Trust assumption 4

The channels on which the voters cast their votes are anonymous.

Without this, coercion-resistance is violated.

Defense:
- another mixing...
  - but this may not be enough...
- physical ballot boxes

Trust assumption 5

At least one of the ballot boxes to which a voter submits his vote is correct.

Trust assumption 6

There exists at least one honest tabulation teller.

Without this coercion-resistance is gone, since then the attacker knows all private tabulation teller keys x_{TT_i} and thus can decrypt all votes and credentials.

Defense: use independent tabulation tellers.

csec
20.7.10
3

Attacks on election authorities

- Malfunctioning RT
  Problem: voter can detect this
  but he cannot prove
  that he didn't get correct
  credentials.

  ⚠ Recreating credentials is a problem
  because the original ones must
  be revoked first. Otherwise
  fairness is lost.

- Non-integrity of a bulletin board.
- Corrupted electoral role. (signed by registrar.)
  → adding missing persons may be easy.
  → detecting fictious voters might be tricky.
  Need an external policy here!

Backdoors in the software.
→ use open source, "own" source

Trust assumption ?

- Decisional Diffie-Hellman is hard.
- RSA ~~and~~ AES are secure.
  or whatever you use instead.

7. Voting specials

ALGORITHM 7.1.

Publicly known: El Gamal parameters (G, q, P).
Input: A message $m \in \mathbb{Z}_q$.
Output: The encoded message $M = \text{encode}(m) \in G$.

1. Return mP

The voting scheme will most of the time encrypt the encoded message. Decoding this — in general — is impossible, but if the message m comes from a known tiny subset of $\mathbb{Z}_q$, we can compute it by brute force. Typical tiny subsets could be the set of indices of the voting options, for example, $\{1, 2, 3, 4, 5, 6\}$ if there are six choices for the voter. Also the possible sum of votes for a certain option may occur, so then the set in question would be $\mathbb{N}_{\leq 2500}$ in a distinct with 2500 voters.

ALGORITHM 7.2. Credential encryption.

Publicly known: El Gamal parameters (G, q, P).
Input: The public key K_{TT} of a tabulation teller, a private credential share $s \in \mathcal{M}$, the temporary private key $t \in \mathbb{Z}_q^\times$ and the identifiers of registration teller rid and voter vid.
Output: credenc(s, t, K_{TT}, rid, vid).

1. Pick a random temporary keys $u \longleftarrow \mathbb{Z}_q$.
2. Encrypt $(T, Y) \leftarrow (tP, \text{encode}(s) + tK_{TT})$.
3. Compute a challenge $c \leftarrow \mathbb{Z}_q(\text{hash}(uP, T, Y, \text{rid}, \text{vid})) \in \mathbb{Z}_q$.
4. Compute the response $r \leftarrow u + ct$ in Z_q.
5. Return (T, Y, c, r)

ALGORITHM 7.3. Credential verification.

Publicly known: El Gamal parameters (G, q, P).
Input: Public credential share $S = (T, Y, c, r)$ and the identifiers of registration teller rid and voter vid.
Output: credverify$(S, \text{rid}, \text{vid})$.

1. Compute $U \leftarrow rP - cT$ and $c' \leftarrow \mathbb{Z}_q(\text{hash}(U, T, Y, \text{rid}, \text{vid})) \in \mathbb{Z}_q$.
2. Return $c' \stackrel{?}{=} c$

8. Further proofs

PROTOCOL 8.1. Reencryption proof (REENCPF).

Public input: A list $C = [(T_i, Y_i)]_i$ of (reencrypted) ciphertexts, a particular ciphertext $\widehat{C} = (T, Y)$, and the recipients' public key X.

Private input to the prover: An index j into the list C and the reencryption randomness t' such that $\widehat{C} = C_j + \mathrm{enc}_X(\mathcal{O}; t')$.

Output to the prover: $\textsc{ReencPf}(j, t') = (\check{s}, \check{t})$

1. The prover performs 2–8.
2.   For all indices i of C do 3–5
3.   She picks random values $s_i, t_i \longleftarrow \mathbb{Z}_q$.
4. $\widetilde{T}_i = s_i(T_i - T) + t_i P$ and
5. $\widetilde{Y}_i = s_i(Y_i - Y) + t_i X$.
6.   The prover computes $c \leftarrow \mathbb{Z}_q(\mathrm{hash}(\widehat{C}, C, [(\widetilde{T}_i, \widetilde{Y}_i)]_i))$.
7.   The prover computes
    $\check{s}_j \leftarrow c - \sum_{i \neq j} s_i$, and for $i \neq j$ let $\check{s}_i \leftarrow s_i$,
    $\check{t}_j \leftarrow t_j - t'(\check{s}_j - s_j)$, and for $i \neq j$ let $\check{t}_i \leftarrow t_i$.
8.   He sends $(\check{s}, \check{t})$. $\xrightarrow{(\check{s}, \check{t})}$
9. The verifier performs 10–15.
10.   He reconstructs $\widetilde{T}$ and $\widetilde{Y}$:
11.   For all indices i of C do 12–13
12. $\widetilde{T}'_i = \check{s}_i(T_i - T) + \check{t}_i P$ and
13. $\widetilde{Y}'_i = \check{s}_i(Y_i - Y) + \check{t}_i X$.
14.   He computes $c' \leftarrow \mathbb{Z}_q(\mathrm{hash}(\widehat{C}, C, [(\widetilde{T}'_i, \widetilde{Y}'_i)]_i))$, and $d' \leftarrow \sum_i \check{s}_i$.
15.   He verifies $c' \stackrel{?}{=} d'$.

Completeness The reconstruction produces identical results for $i \neq j$ since the prover sends his data there. For $i = j$ however we have

$$\begin{aligned}
\widetilde{T}'_j &= \check{s}_j(T_j - T) + \check{t}_j P \\
&= \left(t'\check{s}_j + \check{t}_j\right) P \\
&= \left(t'\check{s}_j + t_j - t'(\check{s}_j - s_j)\right) P \\
&= (t' s_j + t_j) P = s_j(T_j - T) + t_j P = \widetilde{T}_j.
\end{aligned}$$

The computation for $\widetilde{Y}'_j = \widetilde{Y}_j$ is similar (replace T by Y and P by X).

Protocol 8.2. Vote Proof (VotePf).

Public input: Encrypted credential $(T_1, Y_1, c, r) = \mathrm{CredEnc}(s, t, K_{TT}, rid, vid)$, encrypted choice (T_2, Y_2), the prover's public key X.

Private input to the prover: Temporary keys $t_1, t_2 \in \mathbb{Z}_q$ such that $T_i = t_i P$.

1. The prover picks $s_1, s_2 \longleftarrow \mathbb{Z}_q$.
2. The prover computes $c \leftarrow \mathbb{Z}_q(\mathrm{hash}(P, X, T_1, Y_1, T_2, Y_2, s_1 P, s_2 P))$.
3. The prover computes $r_i \leftarrow s_i - ct_i$ in $\mathbb{Z}_q$.
4. He sends (c, r_1, r_2). $\xrightarrow{(c, r_1, r_2)}$
5. The verifier checks $c \stackrel{?}{=} \mathbb{Z}_q(\mathrm{hash}(P, X, T_1, Y_1, T_2, Y_2, r_1 P + cT_1, r_2 P + cT_2))$.

This is merely a parallel execution of two copies of Protocol 4.2, and proves knowledge of the two temporary encryption keys.

9. Main protocols

Protocol 9.1. Plaintext equivalence test (PET).

Public input: Two ciphertexts $C_j = (T_j, Y_j)$, encrpyted with the tabulation tellers' common public key $X_{\mathrm{TT}} = \sum_i X_i$.

Private input to tabulation teller i: The private key share x_i.

Output: $\mathrm{PET}(C_1, C_2)$

1. Tabulation teller i performs 2–6.
2. Pick a randomizer $z_i \in \mathbb{Z}_q$ and compute $\widetilde{T}_i \leftarrow z_i(T_1 - T_2)$, $\widetilde{Y}_i \leftarrow z_i(Y_1 - Y_2)$.
3. Publish a commitment to $(\widetilde{T}_i, \widetilde{Y}_i)$. $\xrightarrow{\mathrm{commit}(\widetilde{T}_i, \widetilde{Y}_i)}$
4. Wait until commitments of all tabulation tellers are available. $\longleftarrow$
5. Publish $(\widetilde{T}_i, \widetilde{Y}_i)$ and a proof of equality of discrete logarithms for $(T_1 - T_2, Y_1 - Y_2, \widetilde{T}_i, \widetilde{Y}_i)$. $\xrightarrow{(\widetilde{T}_i, \widetilde{Y}_i, \mathrm{EqDlogs}(\dots))}$
6. Wait and verify all commitments and proofs. $\longleftarrow$
7. Let $\widetilde{T} \leftarrow \sum_i \widetilde{T}_i$, $\widetilde{Y} \leftarrow \sum_i \widetilde{Y}_i$.
8. All tabulation tellers jointly decrypt $(\widetilde{T}, \widetilde{Y})$: $\longrightarrow$ $\longleftarrow$

$$m' \leftarrow \mathrm{DistDec}(\widetilde{T}, \widetilde{Y}).$$

9. If $m' = \mathcal{O}$ then Return Equal Else Return Unequal .

ALGORITHM 9.2. Atomic mix operation (MIX).

Input: A list $C = [C_i]_i$ of ciphertexts, and a direction $d \in \{\mathsf{In}, \mathsf{Out}\}$.
Output: An anonymized reencryption $M = \mathrm{Mix}(C)$ of C, and a list of commitments.
Private output: r, w, p.

1. Pick a permutation π of the indices of C. (Instead of picking it, you can also compute it such that the reencrypted list M is sorted.)
2. If $d = \mathsf{In}$ then $p \leftarrow \pi^{-1}$ Else $p \leftarrow \pi$.
3. Pick reencryption randomnesses $r_i \longleftarrow \mathbb{Z}_q^\times$ and commitment randomizers $w_i \longleftarrow \mathcal{R}$.
4. Let $M \leftarrow [\mathrm{Reenc}(C_{\pi(i)}; r_i)]_i$.
5. Let $S \leftarrow [\mathrm{Commit}(w_i, p(i))]$.
6. Return M, S .

PROTOCOL 9.3. The anonymizing mix net (MIXNET).

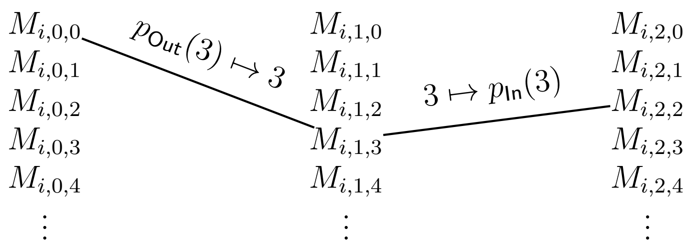
Public input: A list $C = [C_i]_i$ of ciphertexts.
Output: Anonymization $\mathrm{MIXNET}(C)$ of C.

1. Let $M_{0,2} \leftarrow C$.	
2. For $i = 1 \ldots n$ do 3–6	
3. Wait for $M_{i-1,2}$.	$\longleftarrow$
4. Mix i computes $(M_{i,1}, S_{i,1}) \leftarrow \mathrm{Mix}(M_{i-1,2}, \mathsf{Out})$ and publishes that.	$\xrightarrow{M_{i,1},\ S_{i,1}}$
5. Mix i computes $(M_{i,2}, S_{i,2}) \leftarrow \mathrm{Mix}(M_{i,1}, \mathsf{In})$ and publishes that.	$\xrightarrow{M_{i,2},\ S_{i,2}}$
6. Pick a further random value $q_i \longleftarrow \mathcal{R}$ and publish a commitment to it.	$\xrightarrow{\mathrm{Commit}(q_i)}$
7. Wait for all mixes to finish.	$\longleftarrow$
8. Then each mix publishes q_i.	$\xrightarrow{q_i}$
9. Wait and verify all other mixes' commitments.	$\longleftarrow$
10. Let $q \leftarrow \mathrm{hash}(q_1, \ldots, q_n)$.	
11. Compute the challenge $c_i \leftarrow \mathrm{hash}(q, i)$.	
12. For $i \in \{1, \ldots, n\}$ in parallel do 13–20	
13. Mix i publishes r_j or $r_{p(j)}$ depending on $\mathrm{bit}_j(c_i)$, w_j and $p(j)$ from the mixing resulting in $M_{i,1+\mathrm{bit}_j(c_i)}$ for all indices j of C.	$\xrightarrow{\left[\left(\begin{cases} r_j & \text{if } \mathrm{bit}_j(c_i) = 0 \\ r_{p(j)} & \text{if } \mathrm{bit}_j(c_i) = 1 \end{cases}, w_j, p(j)\right)\right]_j}$
14. Now all the mixing information can be erased.	
15. Wait for the other mixes' responses.	$\longleftarrow$
16. Verify $\mathrm{Commit}(w_j, p(j)) = S_{i,1+\mathrm{bit}_j(c_i)}$.	
17. If $\mathrm{bit}_j(c_i) = 0$ then	
18. Verify $\mathrm{Reenc}_X(M_{i-1,2,p(j)}; r_j) = M_{i,1,j}$.	
19. Else	

20. Verify $\text{Reenc}_X(M_{i,1,j}; r_{p(j)}) = M_{i,2,p(j)}$.
21. Return $M_{n,2}$

The q_i-business ensures that the challenges are influenced by all mixes in an unpredictable way. No mix can predetermine its challenge.

The proof of correct mixing reveals exactly half of the mixing process for each index j to the middle layer $M_{i,1}$. In this example:



either the information transforming $M_{i,0,0}$ to $M_{i,1,3}$ or the information transforming $M_{i,1,3}$ to $M_{i,2,2}$ is revealed.

If a mix cheats it remains undetected only with probability $2^{-\#C}$.

Note that these proofs can be checked by anyone after the mixing.

10. The election

Finally, we now reach the election itself.

Note that before the election a supervisor sets up various stuff. In particular a broadcast bulletin board ABB is started and rules for the election are posted there. All verification information will be posted there. Each registration teller generates credentials for each possible voter on its block (precinct), encrypts and posts them to ABB.

We start with the registration.

Protocol 10.1. Registration (Register).

Public input: The distributed public key X_{TT} of the tabulation tellers, a public RSA key K_{RT_i} of the registration teller i. The voter's public designation key X_{vid}. The voter's public registration RSA key K_{vid}. Identifiers of election (eid), voter (vid), registration tellers (rid), and block (bid). Public credentials

$S_j = \mathrm{CredEnc}(s_j; t; X_{TT}; \mathrm{rid}, \mathrm{vid})$ for each registration teller $j \in \mathrm{rid}$.

Private input to registration teller RT_i: Private credential $s_i \in \mathcal{M}$ and encryption randomness $t \in \mathbb{Z}_q^\times$.

Private input to the voter: Private registration RSA key $k_{\mathrm{vid}}, \ldots$

Output to the voter: private credentials $\mathrm{Register}(vid, rid, sid)$

1. The voter picks a nonce N_{vid} and sends the election id eid, his id vid, and the nonce encrypted to the registration teller i. $\xrightarrow{\mathrm{RSAenc}_{K_{\mathrm{RT}_i}}(\mathrm{eid}, \mathrm{vid}, N_{\mathrm{vid}})}$
2. The registration teller RT_i verifies that vid is a voter in block (precinct) bid in election eid, and that for each registration tellers j in rid the public credential S_j is available and $\mathrm{CredVer}(S_j; j, \mathrm{vid})$ succeeds.
3. The registration teller picks a nonce N_R and an AES key k (of security level ℓ).
4. Send the registration teller ids rid, the nonces N_R and N_V and the chosen AES key k to the voter. $\xleftarrow{\mathrm{RSAenc}_{K_{\mathrm{vid}}}(\mathrm{rid}, N_R, N_V, k)}$
5. The voter decrypts and verifies rid and N_V, and sends the nonce N_R back to the registration teller RT_i. $\xrightarrow{N_R}$
6. The registration teller RT_i verifies N_R.
7. The registration teller picks $t' \xleftarrow{\text{🎲}} \mathbb{Z}_q^\times$ and computes $w \leftarrow t' - t$ and another encryption $S_i' \leftarrow \mathrm{Enc}(s_i; t', X_{TT})$ of the private credential.
8. The registration teller sends AES encrypted the private credential share and the new randomness t' together with a designated verifier proof that S_i and S_i' encrypt the same message. $\xleftarrow{\mathrm{AESenc}_k(s_i, t', \mathrm{DVRP}(\ldots), \mathrm{bid})}$
9. The voter decrypts and verifies the designated verifier proof against S_i from the bulletin board.

ALGORITHM 10.2. Fake credentials (FAKECREDENTIAL).

Input obtained from registration: Private credential shares s_i, public credential shares S_i, reencryption factors t_i, and designated verifier proofs D_i from each registration teller RT_i.

Input: Index set L of registration teller for which to fake shares. The voter's designation key pair $(X_{\mathrm{vid}}, x_{\mathrm{vid}})$.

Output: Fake private credential shares ...

1. For i do 2–10
2. If $i \in L$ then
3. Pick $\widetilde{t}_i \longleftarrow \mathbb{Z}_q^\times$.
4. Pick $\widetilde{s}_i$ randomly.
5. $\widetilde{S}_i \leftarrow \mathrm{enc}(\widetilde{s}_i; \widetilde{t}_i; X_{\mathrm{TT}})$.
6. Compute a non-interactive fake designated verifier proof $\widetilde{D}_i$ by Protocol 6.4
7. Else
8. Let $\widetilde{t}_i \leftarrow t_i$.
9. Let $\widetilde{s}_i \leftarrow s_i$.
10. Let $\widetilde{D}_i \leftarrow D_i$
11. Return $[(\widetilde{s}_i, \widetilde{t}_i, \widetilde{D}_i)]_i$

PROTOCOL 10.3. Vote (VOTE).

Public input: The distributed public key X_{TT} of the tabulation tellers. Well-known choice ciphertext list C.

Private input: The voter's choice t and his credentials s.

Output to the ballot box: $\mathrm{Vote}(t, s)$

1. The voter picks a randomness r_s and encrypts his credentials $S \leftarrow \mathrm{enc}(s; r_s; X_{TT})$ for the tabulation tellers.
2. He picks a randomness r_v and reencrypts his choise C_t: $V \leftarrow \mathrm{reenc}(C_t; r_v)$.
3. He prepares a vote proof P_w of correct voting by Protocol 8.2 with inputs S, V, r_s, r_v, and further context.
4. He prepares a REENCPF P_k that V is a reencryption of one of the cipher texts C by Protocol 8.1.
5. Let vote $\leftarrow (S, V, P_w, P_k)$ and send this to the ballot box. $\xrightarrow{\text{vote}}$

Protocol 10.4. Tabulate (Tabulate).

Principals: Tabulation tellers $\mathrm{TT}_1, \ldots, \mathrm{TT}_n$, broadcast bulletin board ABB, ballot boxes $\mathrm{VBB}_1, \ldots, \mathrm{VBB}_m$, supervisor Sup.
Public input: X_{TT}, contents of bulletin board ABB.
Private input to TT_i: Private key share x_i of X_{TT}.
Output: Election tally for one block.

1. Each ballot box VBB_i posts commitments on the list of all votes on the tabulation board ABB.	$\xrightarrow{\text{Commit(received votes)}}$
2. The supervisor signs the list of all received VBB commitments.	$\xrightarrow{\mathrm{sign}_{\mathrm{Sup}}(\text{ABB so far})}$
3. The tabulation tellers TT_i jointly execute 4–11.	
4. **Retrieve votes**. Retrieve all votes from all endorsed ballot boxes VBB_i. Verify the commitments. Let $A \leftarrow$ list of votes.	$\xleftarrow{\text{votes}}$ $\xrightarrow{A}$
5. **Check proofs**. Verify all VotePfs and ReencPfs in retrieved votes. Eliminate any votes with an invalid proof. Let B be the list of remaining votes.	$\xrightarrow{B}$
6. **Duplicate elimination**. Run the plaintext equivalence test $\mathrm{PET}(S'_i, S'_j)$ for all pairs (i, j), where S'_x is the encrypted credential in vote B_x. Eliminate equivalent votes according to a revoting policy. Let C be the list of remaining votes.	$\xrightarrow{C}$
7. **Mix votes**. $D \leftarrow \mathrm{MixNet}(C)$.	$\xrightarrow{D}$
8. **Mix credentials**. Let E be the list of all initially created encrypted credentials. Anonymize it: $F \leftarrow \mathrm{MixNet}(E)$.	$\xleftarrow{E}$ $\xrightarrow{F}$
9. **Invalid elimination**. Run the plaintext equivalence test $\mathrm{PET}(S_i, S'_j)$ for all pairs (i, j) where $S_i = F_i$, $S_j = D_j$. Eliminate votes from D for which there is no equivalent credential found in F. Let G be the list of remaining votes.	$\xrightarrow{G}$
10. **Decrypt**. Let $H_i \leftarrow \mathrm{DistDec}(G_i)$ for all i.	$\xrightarrow{H}$
11. **Tally**. Compute the tally of H according to an election method specified by the supervisor.	$\xrightarrow{\text{tally}}$
12. Finally, the supervisor endorses the tally (if ...).	$\xrightarrow{\text{Sign ABB so far.}}$

11. Security model and trust assumptions

...

References

MICHAEL R. CLARKSON, STEPHEN CHONG & ANDREW C. MYERS (2008). Civitas: Toward a Secure Voting System. Computing and Information Science Technical Report `http://hdl.handle.net/1813/7875`, Department of Computer Science, Cornell University. Previously TR 2007-2081.

AMOS FIAT & ADI SHAMIR (1986). How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology: Proceedings of CRYPTO '86,* Santa Barbara CA, A. M. ODLYZKO, editor, number 263 in Lecture Notes in Computer Science, 186–194. Springer-Verlag. ISSN 0302-9743.