Norwegian University of Science and Technology

PROTOCOL COMPOSITION 2: DLOG

TTM4205 – Lecture 16

Tjerand Silde

05.11.2024

Contents

Contents

OpenPGP

- Protocol for securing email.
- Standardized in RFC4880.
- Encryption: ElGamal Hybrid Encryption (...or RSA).
- Signatures: DSA or RSA.

We will look at a cross-implementation attack on OpenPGP.

On the (in)security of ElGamal in OpenPGP

Luca De Feo*
IBM Research Europe – Zurich
Rüschlikon, Switzerland

Bertram Poettering*
IBM Research Europe – Zurich
Rüschlikon, Switzerland

Alessandro Sorniotti*
IBM Research Europe – Zurich
Rüschlikon, Switzerland

Figure: https://eprint.iacr.org/2021/923

ElGamal Hybrid Encryption

Let $\mathbb{G}$ be a group. The ElGamal hybrid encryption scheme works as follows:

KGen : Sample secret key sk and publish the public key $\mathsf{pk} = g^{\mathsf{sk}}$.

Enc : Sample uniform x, compute $X = g^x$, and use $k = \mathsf{H}(\mathsf{pk}^x)$ as a secret key for AES to encrypt message m as ctx. Send (X, ctx).

Dec : On receiving the ciphertext (X, ctx), compute the AES key as $k = \mathsf{H}(X^{\mathsf{sk}})$ and decrypt ctx to get the message m.

ElGamal Hybrid Encryption

Key Generation Questions

- What kind of group should $\mathbb{G}$ be?
- How should the element g be selected?
- Which interval should sk and x be sampled from?

We will have a look at four different configurations that are all used in practice. In all cases, $\mathbb{G}$ is the multiplicative group $\mathbb{Z}_p^\times$ for some prime p.

Two Simple Configurations

Configuration A

- $\mathbb{G} = \mathbb{Z}_p^\times$ where $p - 1$ has at least one large prime factor q.
- The element g is a generator of the group $\mathbb{G}$.
- sk and x are sampled from the interval $[0, p - 1]$.

Two Simple Configurations

Configuration A

- $\mathbb{G} = \mathbb{Z}_p^\times$ where $p - 1$ has at least one large prime factor q.
- The element g is a generator of the group $\mathbb{G}$.
- sk and x are sampled from the interval $[0, p - 1]$.

Configuration B

- $\mathbb{G} = \mathbb{Z}_p^\times$ where $p - 1$ has at least one large prime factor q.
- The element g is a generator of the subgroup $\mathbb{G}' \subseteq \mathbb{G}$ of order q
- sk and x are sampled from $[0, q - 1]$ for efficiency.

Note that in Configuration B, we have that $q \ll p$.

Two more Configurations

Configuration C (Safe Primes)

- $\mathbb{G} = \mathbb{Z}_p^\times$ where $p - 1 = 2q$, where q is prime.
- $g = 4$ (always a generator of the group $\mathbb{G}' \subseteq \mathbb{G}$ of order q)
- sk and x are sampled from the interval $[0, p - 1]$.

Two more Configurations

Configuration C (Safe Primes)

- $\mathbb{G} = \mathbb{Z}_p^\times$ where $p - 1 = 2q$, where q is prime.
- $g = 4$ (always a generator of the group $\mathbb{G}' \subseteq \mathbb{G}$ of order q)
- sk and x are sampled from the interval $[0, p - 1]$.

Configuration D (Lim-Lee Primes)

- $\mathbb{G} = \mathbb{Z}_p^\times$ where $p - 1 = 2 \cdot q_1 \cdot q_2 \cdots q_n$, with q_i same sized primes.
- The element g is a generator of the subgroup $\mathbb{G}' \subseteq \mathbb{G}$ of some order q_i
- sk and x are sampled from $[0, q_i - 1]$ for efficiency.

Contents

Algorithms for Discrete Logarithms

Pohlig-Hellman

Let $\mathbb{G}$ be generated by a generator g where $|\mathbb{G}| = p_1^{e_1} \cdot p_2^{e_2} \cdots p_n^{e_n}$ and p_i are prime numbers. Given $X \in \mathbb{G}$, we want to find $x \in \mathbb{Z}_p$ such that $g^x = X$.

The Pohlig-Hellman algorithm reduces this to the task of computing discrete logarithms in subgroups of order p_i, and then combining the results using the Chinese remainder theorem to find the value $x \in \mathbb{Z}_p$ of X in $\mathbb{G}$.

Algorithms for Discrete Logarithms

Pollard's Kangaroo

Let $\mathbb{G}$ be generated by a generator g where $|\mathbb{G}| = q$ and q is a prime number. Given $X \in \mathbb{G}$, we want to find $x \in [a, b]$ such that $g^x = X$.

The Pollard's Kangaroo algorithm can compute the discrete logarithm $x \in [a, b]$ of X in time and space roughly $\sqrt{b - a}$ given known integers a and b.

Contents

Cross-Implementation Attack on ElGamal

- By themselves, configuration A/B/C/D are all secure.

Cross-Implementation Attack on ElGamal

- By themselves, configuration A/B/C/D are all secure.
- However, by combining them, they can become insecure.

Cross-Implementation Attack on ElGamal

- By themselves, configuration A/B/C/D are all secure.
- However, by combining them, they can become insecure.
- Attack: A user using configuration B (small secret mod q) sends a ciphertext to someone using configuration A (large secret mod p):

Cross-Implementation Attack on ElGamal

- By themselves, configuration A/B/C/D are all secure.
- However, by combining them, they can become insecure.
- Attack: A user using configuration B (small secret mod q) sends a ciphertext to someone using configuration A (large secret mod p):
  - The sender uses "small" a exponent $x \in [0, \ldots, \approx 2^{256}]$.
  - The receiver uses generator g of group $\mathbb{G}$ of order $p - 1 = f_1^{e_1} \cdot f_2^{e_2} \cdots f_n^{e_n} \cdot q$, where f_i are small enough primes to solve discrete logarithms but q is large.

Cross-Implementation Attack on ElGamal

The attack works as following:

1. Use the Pollard's Kangaroo algorithm to solve the discrete logarithm modulo each of the small primes f_i.
2. Use the Pohlig-Hellman algorithm to combine the solutions modulo $M = f_1^{e_1} \cdot f_2^{e_2} \cdots f_n^{e_n}$ as $w \equiv x \pmod{M}$ where $p - 1 = M \cdot q$ for a large q.
3. Note now that $X = g^{z \cdot M + w}$ for some unknown $z \in [0, \ldots, q/M]$.
4. Finally we find z by computing the discrete logarithm of X/g^w to the base g^M using Pollard's Kangaroo algorithm again.

This attach also works for case D with $p - 1 = f_1^{e_1} \cdots f_n^{e_n} \cdot q_1 \cdots q_\ell$ where the secret is modulo q_i and we can retrieve it using smaller $M = f_1^{e_1} \cdot f_2^{e_2} \cdots f_n^{e_n}$.

Contents

A triple-ElGamal (encryption)

In the original scheme, there is a **multi-level** variant:
Choose $p_1 < p_2 < p_3$, three safe primes, together with 3 generators g_1, g_2, g_3.

The **keys** are

$$\mathrm{sk} = (\mathrm{sk}_1, \mathrm{sk}_2, \mathrm{sk}_3); \quad \mathrm{pk} = (g_1^{\mathrm{sk}_1}, g_2^{\mathrm{sk}_2}, g_3^{\mathrm{sk}_3}),$$

The **encryption** of a message $m \in \mathbb{Z}/p_1\mathbb{Z}$ is obtained by

$$\begin{aligned}
(a_1, b_1) &:= \mathsf{Enc}_{g_1,\mathrm{pk}_1}(m); \quad \text{map } a_1 \text{ to } \mathbb{Z}/p_2\mathbb{Z};\\
(a_2, b_2) &:= \mathsf{Enc}_{g_2,\mathrm{pk}_2}(a_1); \quad \text{map } a_2 \text{ to } \mathbb{Z}/p_3\mathbb{Z};\\
(a_3, b_3) &:= \mathsf{Enc}_{g_3,\mathrm{pk}_3}(a_2),
\end{aligned}$$

and the encrypted message is

$$\mathsf{MultiEnc}(m) = (b_1, b_2, a_3, b_3).$$

Rem. All mapping are obtained by canonical lifting to $\mathbb{Z}$.

A triple-ElGamal (decryption)

Knowing sk, the operations can be reversed to **decrypt** m from (b_1, b_2, a_3, b_3):

$$
\begin{aligned}
a_2 &:= \mathrm{Dec}_{g_3,\mathrm{sk}_3}(a_3, b_3); \quad \text{map } a_2 \text{ to } \mathbb{Z}/p_2\mathbb{Z}; \\
a_1 &:= \mathrm{Dec}_{g_2,\mathrm{sk}_2}(a_2, b_2); \quad \text{map } a_1 \text{ to } \mathbb{Z}/p_1\mathbb{Z}; \\
m &:= \mathrm{Dec}_{g_1,\mathrm{sk}_1}(a_1, b_1).
\end{aligned}
$$

Due to the inequality $p_1 < p_2 < p_3$, **this works**.

A triple-ElGamal (decryption)

Knowing sk, the operations can be reversed to **decrypt** m from (b_1, b_2, a_3, b_3):

$$\begin{aligned} a_2 &:= \mathrm{Dec}_{g_3,\mathrm{sk}_3}(a_3, b_3); \quad \text{map } a_2 \text{ to } \mathbb{Z}/p_2\mathbb{Z}; \\ a_1 &:= \mathrm{Dec}_{g_2,\mathrm{sk}_2}(a_2, b_2); \quad \text{map } a_1 \text{ to } \mathbb{Z}/p_1\mathbb{Z}; \\ m &:= \mathrm{Dec}_{g_1,\mathrm{sk}_1}(a_1, b_1). \end{aligned}$$

Due to the inequality $p_1 < p_2 < p_3$, **this works**.

Security.
Contrary to triple-DES where the number of operations to break the system is squared, here it is just multiplied by 3.

Breaking the scheme is not harder than to

break the 3 underlying ElGamal independently.

DLP with CADO-NFS

Running times on my 4-year old nothing-special desk PC:

key number	time
sk_1	425 sec
sk_2	507 sec
sk_3	314 sec

Each line includes 2 runs of CADO-NFS (one for g_i, one for pk_i); but many steps are (automatically) shared.

Figure: They used 256-bit finite field ElGamal...
https://rwc.iacr.org/2020/slides/Gaudry.pdf

Contents

Three Lessons From Threema: Analysis of a Secure Messenger

Kenneth G. Paterson
Applied Cryptography Group, ETH Zurich

Matteo Scarlata
Applied Cryptography Group, ETH Zurich

Kien Tuong Truong
Applied Cryptography Group, ETH Zurich

Figure: `https://breakingthe3ma.app/files/Threema-PST22.pdf`

Bird's Eye View of the Threema Protocol

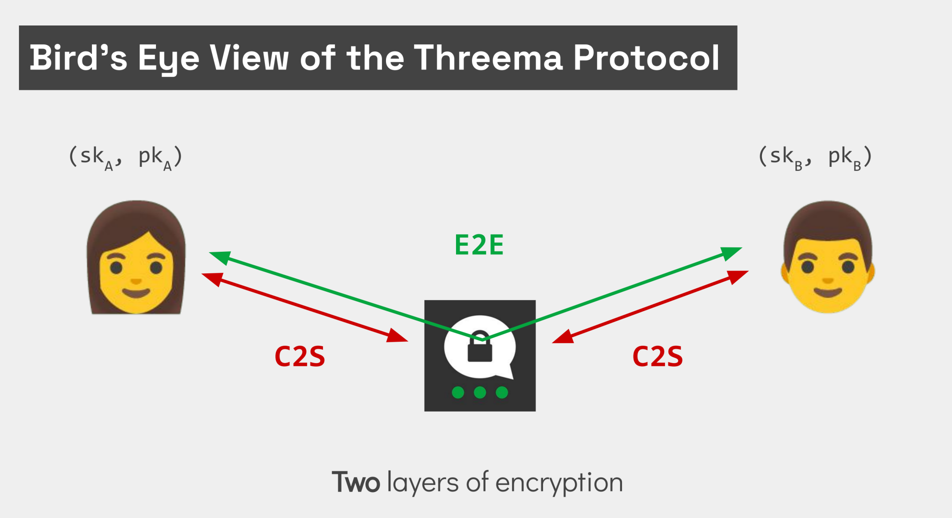
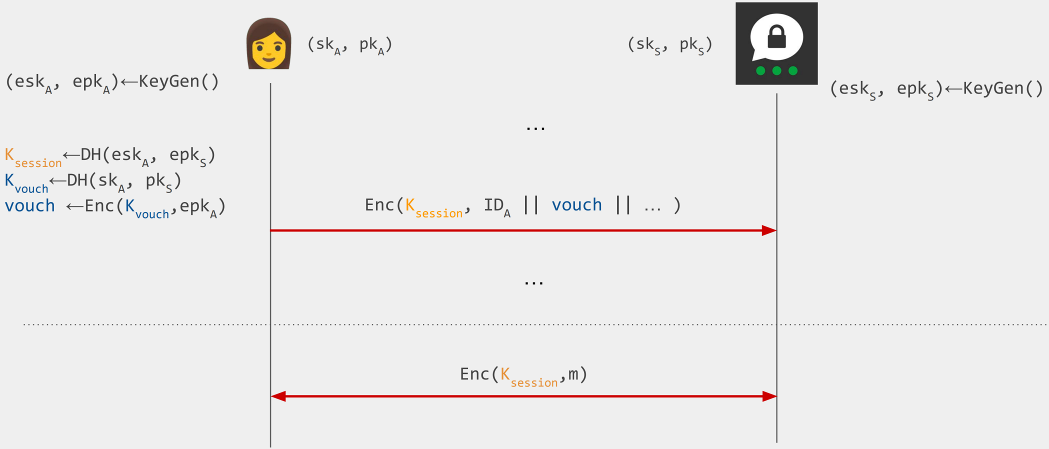


Figure: https://iacr.org/submit/files/slides/2023/rwc/rwc2023/75/slides.pdf

The C2S Protocol*



* Simplified, details omitted

The C2S Protocol: Vouch Box

$K_{vouch} \leftarrow DH(sk_A,\ pk_S)$ DH(long-term, long-term)

$vouch \leftarrow Enc(K_{vouch}, epk_A)$ Enc(some value)

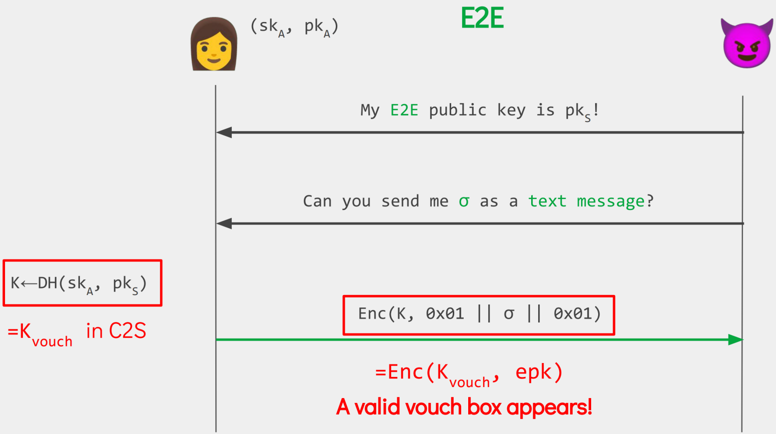
What if we could find a special keypair `(esk, epk)` such that:

`epk = 0x01 || σ || 0x01`

UTF-8 valid string of 30B

NTNU | Norwegian University of Science and Technology

Attacking the C2S Protocol



Part 1: Getting That Key

```
epk = 0x01 || σ || 0x01
```

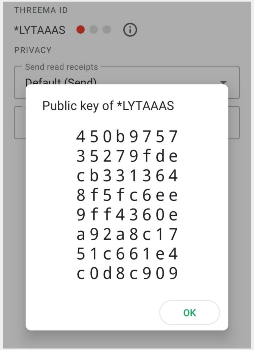
UTF-8 valid string of 30B

Requires sampling 2^{51} keys!

Part 2: The Bamboozling

- Threema Gateway: paid API
- Can register accounts **with arbitrary public keys**
- **Without proof of possession** of the corresponding private key!

=> ***LYTAAAS**



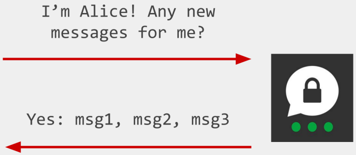

```
public static final byte[] SERVER_PUBKEY = new byte[] {
    (byte) 0x45, (byte) 0x0b, (byte) 0x97, (byte) 0x57,
    (byte) 0x35, (byte) 0x27, (byte) 0x9f, (byte) 0xde,
    (byte) 0xcb, (byte) 0x33, (byte) 0x13, (byte) 0x64,
    (byte) 0x8f, (byte) 0x5f, (byte) 0xc6, (byte) 0xee,
    (byte) 0x9f, (byte) 0xf4, (byte) 0x36, (byte) 0x0e,
    (byte) 0xa9, (byte) 0x2a, (byte) 0x8c, (byte) 0x17,
    (byte) 0x51, (byte) 0xc6, (byte) 0x61, (byte) 0xe4,
    (byte) 0xc0, (byte) 0xd8, (byte) 0xc9, (byte) 0x09
};
```

Vouch Box Forgery

- **C2S x E2E** cross-protocol attack:
- Sending a text message…

  compromises client

  authentication **forever**!



Attack: Vouch Box Forgery

Contents

Mesh Messaging in Large-scale Protests: Breaking Bridgefy

Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková

Royal Holloway, University of London
{martin.albrecht,jorge.blascoalis,rikke.jensen,lenka.marekova}@rhul.ac.uk

Figure: https://eprint.iacr.org/2021/214.pdf

Bridgefy (Again)

Breaking Bridgefy, again:
Adopting libsignal is not enough

Martin R. Albrecht
Information Security Group,
Royal Holloway, University of London

Raphael Eikenberg
Applied Cryptography Group,
ETH Zurich

Kenneth G. Paterson
Applied Cryptography Group,
ETH Zurich

Figure: `https://www.usenix.org/system/files/sec22fall_albrecht.pdf`

Practically-exploitable Vulnerabilities in the Jitsi Video Conferencing System

Robertas Maleckas
ETH Zürich
Switzerland
robertas.maleckas@alumni.ethz.ch

Kenneth G. Paterson
ETH Zürich
Switzerland
kenny.paterson@inf.ethz.ch

Martin R. Albrecht
King's College London
UK
martin.albrecht@kcl.ac.uk

Figure: `https://eprint.iacr.org/2023/1118.pdf`

Practically-exploitable Cryptographic Vulnerabilities in Matrix

Martin R. Albrecht[*], Sofía Celi[†], Benjamin Dowling[‡] and Daniel Jones[§]
[*] King's College London, martin.albrecht@kcl.ac.uk
[†] Brave Software, cherenkov@riseup.net
[‡] Security of Advanced Systems Group, University of Sheffield, b.dowling@sheffield.ac.uk
[§] Information Security Group, Royal Holloway, University of London, dan.jones@rhul.ac.uk

Figure: https://nebuchadnezzar-megolm.github.io/static/paper.pdf

Contents

Conclusions

Conclusions

- End-to-end security is hard

Conclusions

- End-to-end security is hard
- Composing protocols is hard

Conclusions

- End-to-end security is hard
- Composing protocols is hard
- Have very clear descriptions

Conclusions

- End-to-end security is hard
- Composing protocols is hard
- Have very clear descriptions
- Always (try to) prove security

Conclusions

- End-to-end security is hard
- Composing protocols is hard
- Have very clear descriptions
- Always (try to) prove security
- Use up-to-date modern primitives

Conclusions

- End-to-end security is hard
- Composing protocols is hard
- Have very clear descriptions
- Always (try to) prove security
- Use up-to-date modern primitives
- Be careful about reusing primitives

Conclusions

- End-to-end security is hard
- Composing protocols is hard
- Have very clear descriptions
- Always (try to) prove security
- Use up-to-date modern primitives
- Be careful about reusing primitives
- Authenticate all messages and metadata

Conclusions

- End-to-end security is hard
- Composing protocols is hard
- Have very clear descriptions
- Always (try to) prove security
- Use up-to-date modern primitives
- Be careful about reusing primitives
- Authenticate all messages and metadata
- Always use ephemeral keys for sessions

Conclusions

The Signal Protocol and TLS 1.3 are two out of few protocols that we got right.
It took many years of research, analysis, attacks and experience to get it right.

Questions?