Puncturable Encryption – A Fine-Grained Approach to Forward Security and More

Christoph Striecks
joint work with Sebastian Ramacher & Daniel Slamanig

April 15, 2022

AIT Austrian Institute of Technology

Signatures

Encryption

Key Exchange

Identification

MACs

Security and Trust of Cryptographic Keys

Advanced Primitives

Secure Computation

Zero-Knowledge Proofs

The Heartbleed Bug

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

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

What leaks in practice?

We have tested some of our own services from attacker's perspective. We attacked ourselves from outside, without leaving a trace. Without using any privileged information or credentials we were able steal from ourselves the secret keys used for our X.509 certificates, user names and passwords, instant messages, emails and business critical documents and communication.

How to stop the leak?

As long as the vulnerable version of OpenSSL is in use it can be abused. Fixed OpenSSL has been released and now it has to be deployed. Operating system vendors and distribution, appliance vendors, independent software vendors have to adopt the fix and notify their users. Service providers and users have to install the fix as it becomes available for the operating systems, networked appliances and software they use.

Q&A

What is the CVE-2014-0160?

CVE-2014-0160 is the official reference to this bug. CVE (Common Vulnerabilities and Exposures) is the Standard for Information Security Vulnerability Names maintained by MITRE. Due to co-incident discovery a duplicate CVE, CVE-2014-0346, which was assigned to us, should not be used, since others independently went public with the CVE-2014-0160 identifier.

Why it is called the Heartbleed Bug?

Bug is in the OpenSSL's implementation of the TLS/DTLS (transport layer security protocols) heartbeat extension (RFC6520). When it is exploited it leads to the leak of memory contents from the server to the client and from the client to the server.

An extended abstract of this article appears in the proceedings of CCS 2021. This is the full version and available in the IACR eprint archive.

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

ABSTRACT

Roughly four decades ago, Taher ElGamal put forward what is today one of the most widely known and best understood public key encryption schemes. ElGamal encryption has been used in many different contexts, chiefly among them by the OpenPGP standard. Despite its simplicity, or perhaps because of it, in reality there is a large degree of ambiguity on several key aspects of the cipher. Each library in the OpenPGP ecosystem seems to have implemented a slightly different "flavour" of ElGamal encryption. While –taken in isolation– each implementation may be secure, we reveal that in the interoperable world of OpenPGP, unforeseen *cross-configuration attacks* become possible. Concretely, we propose different such attacks and show their practical efficacy by recovering plaintexts and even secret keys.

CCS CONCEPTS

• **Security and privacy → Public key encryption**.

KEYWORDS

OpenPGP, ElGamal encryption, side-channel attacks, key-recovery attacks, modular exponentiation

1 INTRODUCTION

The ElGamal cryptosystem [14] is one of the oldest and best-known public key encryption schemes. In the 80's and 90's it earned wide adoption for being simultaneously efficient and patent-free. Its most prominent use is arguably as part of OpenPGP [12], a standard aimed to promote consumable, interoperable email security, where it has been the default and most popular encryption option for decades [30]. While the change in patent status of RSA encryption slightly reduced its popularity, at the time of writing, still at least 1 in 6 registered OpenPGP keys have an ElGamal subkey [3], with about a 1,000 new registrations per year.

The ElGamal scheme builds on elegant mathematical structures and can be defined very compactly. This simplicity, together with the opportunity to mature for roughly four decades now, suggests that a crisp specification with clear parameter choices, rules, and algorithms would be present in international standards, in particular in OpenPGP. Surprisingly, this turns out *not* to be the case: our research reveals that OpenPGP's understanding of ElGamal encryption is open to interpretation, with several choices subject to the discretion of the implementer.

In this article we consider *cross-configuration attacks* on OpenPGP. Such attacks emerge when different interpretations ('configurations') of the same standard interact insecurely with each other. Towards identifying such conditions for ElGamal encryption, we need to first understand the universe of OpenPGP interpretations that are used in practice. We approach this challenge from various angles: we carefully study RFC4880 [12] which defines OpenPGP, we inspect the source code of three relevant OpenPGP-implementing software libraries (the `Go` standard library, `Crypto++`, and `gcrypt`), and we conduct a large-scale examination of millions of keys registered on OpenPGP key servers.

Our results reveal an insecure posture. For instance, we develop and prototype a plaintext recovery attack that can be mounted on ciphertexts produced by the ubiquitous GNU Privacy Guard (and other implementations, e.g. `Crypto++`) against keys generated following the original ElGamal specification [14]. The attack is effective against 2048 bit keys, which are considered secure at the time of writing. Our OpenPGP key server analysis reveals that more than 2,000 OpenPGP users are currently exposed.[1] We further illustrate how cross-configuration attacks can be combined with known side-channel exploitation techniques like FLUSH+RELOAD [38] or PRIME+PROBE [33]. One of our targets is the ElGamal implementation of `gcrypt`, the cryptographic library used by the GNU Privacy Guard. Interestingly, `gcrypt` has already been fixed twice after seminal work [21, 38] on side-channel attacks identified weaknesses. Concretely, by conducting an end-to-end attack we show that if a 2048 bit ElGamal key generated by `Crypto++` is used by `gcrypt` to decrypt a ciphertext, then an attacker that is OS- or VM-colocated with the decrypter can fully recover the decryption key.

Given that interoperability is the explicit and almost exclusive goal of any standardization effort, and commonplace in the OpenPGP world, we conclude that our attack conditions are as realistic as the attack results awakening. Our research is timely since a new version of the OpenPGP standard is currently being discussed [18]; we hope that our findings will influence that discussion.

This manuscript is organised as follows: In Section 2 we survey (a) the meaningful options available when implementing ElGamal encryption, (b) the options adopted by the `Go`, `Crypto++`, and `gcrypt` libraries, and (c) the options picked by over 800,000 users in practice (as far as reflected on key server databases); we also report on further interesting findings from our key server crawl. In Section 3 we recall various standard algorithms for solving discrete logarithms. In Section 4 we describe "vanilla" cross-configuration attacks, and in Section 5 we describe those combined with side-channel attacks. In Section 6 we conduct end-to-end exploits and describe how we bring in the required side-channel information. We conclude in Section 7.

1.1 Related Work

Since ElGamal encryption was first proposed [14], research efforts were both steered towards formally confirming its security (e.g. via reductions to the DDH problem [34]) and to shed light on its insecurities (e.g. when used in its textbook form [10]). CVE-2018-6829 [2]

*{fde,poe,aso}@zurich.ibm.com

[1] We found that at most a small fraction of ElGamal keys is formed according to the original specification of [14]; otherwise, many OpenPGP users would be affected.

Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design

Alon Shakevsky
shakevsky@mail.tau.ac.il

Eyal Ronen
eyal.ronen@cs.tau.ac.il

Avishai Wool
yash@eng.tau.ac.il

Tel-Aviv University

Abstract

ARM-based Android smartphones rely on the TrustZone hardware support for a Trusted Execution Environment (TEE) to implement security-sensitive functions. The TEE runs a separate, isolated, TrustZone Operating System (TZOS), in parallel to Android. The implementation of the cryptographic functions within the TZOS is left to the device vendors, who create proprietary undocumented designs.

In this work, we expose the cryptographic design and implementation of Android's Hardware-Backed Keystore in Samsung's Galaxy S8, S9, S10, S20, and S21 flagship devices. We reversed-engineered and provide a detailed description of the cryptographic design and code structure, and we unveil severe design flaws. We present an IV reuse attack on AES-GCM that allows an attacker to extract hardware-protected key material, and a downgrade attack that makes even the latest Samsung devices vulnerable to the IV reuse attack. We demonstrate working key extraction attacks on the latest devices. We also show the implications of our attacks on two higher-level cryptographic protocols between the TrustZone and a remote server: we demonstrate a working FIDO2 WebAuthn login bypass and a compromise of Google's Secure Key Import.

We discuss multiple flaws in the design flow of TrustZone based protocols. Although our specific attacks only apply to the ≈100 million devices made by Samsung, it raises the much more general requirement for open and proven standards for critical cryptographic and security designs.

1 Introduction

Beyond their usage in many and various daily activities, smartphones are increasingly used for many security-critical tasks, such as the protection of sensitive data (messages, images, files), cryptographic key management [26], FIDO2 web authentication [69], Digital Rights Management [68] (DRM),

Simultaneously, smartphones are becoming more and more complex and present an increasingly larger attack surface. The result is that they have become a major target for malware and malicious attackers. There have been many public exploits that allow an attacker to escalate privileges in the Android OS, gaining execution as root or even as the OS kernel [9, 14, 20, 21, 43]. Ideally, such attacks should not be able to compromise the devices' security-critical tasks.

Trusted Execution Environments (TEEs) are largely used in modern mobile devices to provide an isolated environment for execution of Trusted Applications (TAs) that can securely perform security-critical tasks. They have a relatively small codebase and limited APIs.

In contrast, the Rich Execution Environments (REEs), such as Android OS, cannot be fully audited and trusted (due to their complexity). An isolated TEE can be used alongside the REE to implement security-sensitive functions. This makes it harder for an attacker to compromise these functions, as the attack surface is significantly reduced and is limited to communication with the TEE.

In other words, the goal of the TEE is to withstand attacks from a fully compromised REE, including by privileged adversaries with kernel or root capabilities.

ARM is the most widely used processor in the mobile and embedded markets [50], and it provides TEE hardware support with ARM TrustZone [3, 8]. TrustZone separates the device into two execution environments:

1. A non-secure REE where the "Normal World" operating system runs.
2. A secure TEE where the "Secure World" operating system runs.

The REE and TEE use separate resources (e.g., memory, peripherals), and the hardware enforces the protection of Secure World.

In most mobile devices, the Android OS runs the non-

naked security by SOPHOS

PRODUCTS › FREE TOOLS › FREE SOPHOS HOME ›

Have you listened to our podcast? Listen now

Adobe security team posts public key – together with private key

23 SEP 2017 5

Cryptography

× Don't show me this again

Get the latest security news in your inbox.

you@example.com Subscribe

I will not make my PRIVATE key PUBLIC.
I will not make my PRIVATE key PUBLIC.
I will not make my PRIVATE key PUBLIC.
I will not make my PRIVATE key PUBLIC.
I will not make my PRIVATE key PUBLIC.
I will not make my PRIVATE key PUBLIC.
I will not make my PRIVATE key PUBLIC.
I will not make my PRIVATE key PUBLIC.
I will not make my PRIVATE key PUBLIC.
I will not make my PRIVATE key PUBLIC.

Previous: Tracking phones without a warrant ruled unc... Next: Monday review – Adobe botches, Apache bleeds a...

by Paul Ducklin

Finnish security researcher Juho Nurminen is a bit of a retweet celebrity right now, for all the wrong reasons.

Not *his* wrong reasons, but the wrong reasons of Adobe's Product Security Incident Response Team (PSIRT).

To explain.

THE A.V. CLUB DEADSPIN GIZMODO JALOPNIK JEZEBEL KOTAKU LIFEHACKER THE ROOT THE TAKEOUT THE ONION THE INVENTORY

GIZMODO Tech. Science. Culture. Shop Subscribe

HOME LATEST TECH NEWS REVIEWS HOW TO SCIENCE EARTHER IO9 EN ESPAÑOL

PRIVACY AND SECURITY

Amazon Engineer Leaked Private Encryption Keys. Outside Analysts Discovered Them in Minutes

By Dell Cameron | 1/22/20 12:24PM | Comments (16)

Photo: Getty

An Amazon Web Services (AWS) engineer last week inadvertently made public almost a gigabyte's worth of sensitive data, including their own personal documents as well as passwords and cryptographic keys to various AWS environments.

While these kinds of leaks are not unusual or special, what is noteworthy here is how quickly the employee's credentials were recovered by a third party, who—to the employee's good fortune, perhaps—immediately warned the company.

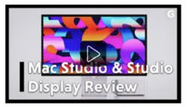
On the morning of January 13, an AWS employee, identified as a DevOps Cloud Engineer on LinkedIn, committed nearly a gigabyte's worth of data to a personal GitHub repository bearing their own name. Roughly 30 minutes later, Greg Pollock, vice president of product at UpGuard, a California-based security

Featured Videos

Russian Court Says Meta Is "Extremist Organization"

Mac Studio and Studio Display Review

Friday 3:15PM



io9 Exclusive Star Trek: Discovery Season 4 Finale Clip: "The Status"

Wednesday 3:15PM

https://nakedsecurity.sophos.com/2017/09/23/adobe-security-team-posts-public-key-together-with-private-key; https://gizmodo.com/amazon-engineer-leaked-private-encryption-keys-outside-1841160934

GitGuardian BLOG VISIT WEBSITE CONTACT SALES DEVSECOPS GLOSSARY

BEST PRACTICES

Exposing secrets on GitHub: What to do after leaking credentials and API keys

If you have discovered that you have just exposed a sensitive file or secrets to a public git repository, there are some very important steps to follow.

MACKENZIE JACKSON
24 MAR 2020 · 6 MIN READ

Share

https://blog.gitguardian.com/leaking-secrets-on-github-what-to-do

How to increase the Resilience of Cryptographic Keys?

Resilience after Key Leakage: Forward Security

- Key leakage: severe issue for key establishment – all data immediately in danger
- Mitigation: forward security – old data still safe
- Efficient in interactive key-exchange protocols and mandatory in TLS 1.3 with 1-RTT
- 99% of Internet sites surveyed by Qualys SSL Labs support it
- Highly recognized by industry: Google, Apple, Meta, Microsoft, Cloudflare, ...

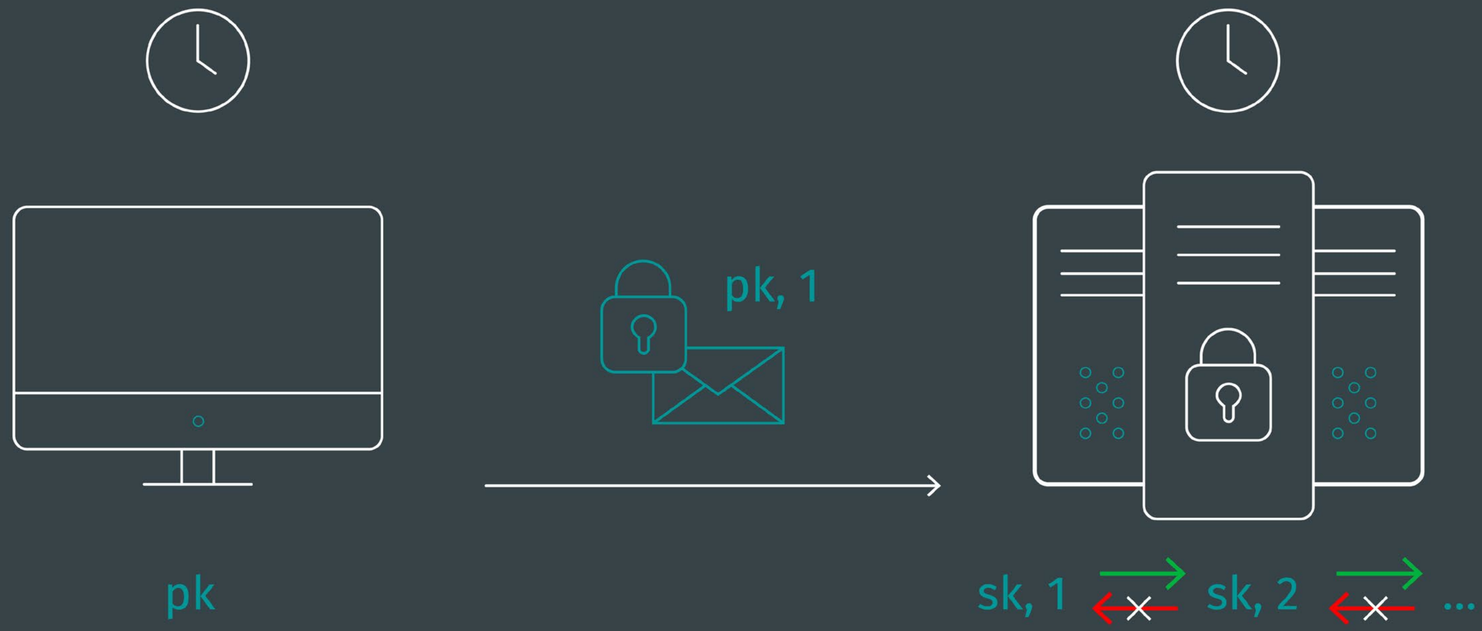
However: much harder in non-interactive settings!

Non-Interactive Forward Security

- Requirements: long-term fixed public key, (minimum to) no state between entities
- Particularly the case in modern distributed settings with many decentralized entities
- Deployed by, e.g., Dfinity in their non-interactive distributed key generation and key resharing protocol

How to even achieve this?

Non-Interactive Forward Security



Epoch-based synchronization [Canetti-Halevi-Katz 2003], compact parameter sizes (independent on # of epochs), coarse-grained

Fine-Grained Forward Security: Puncturable Encryption

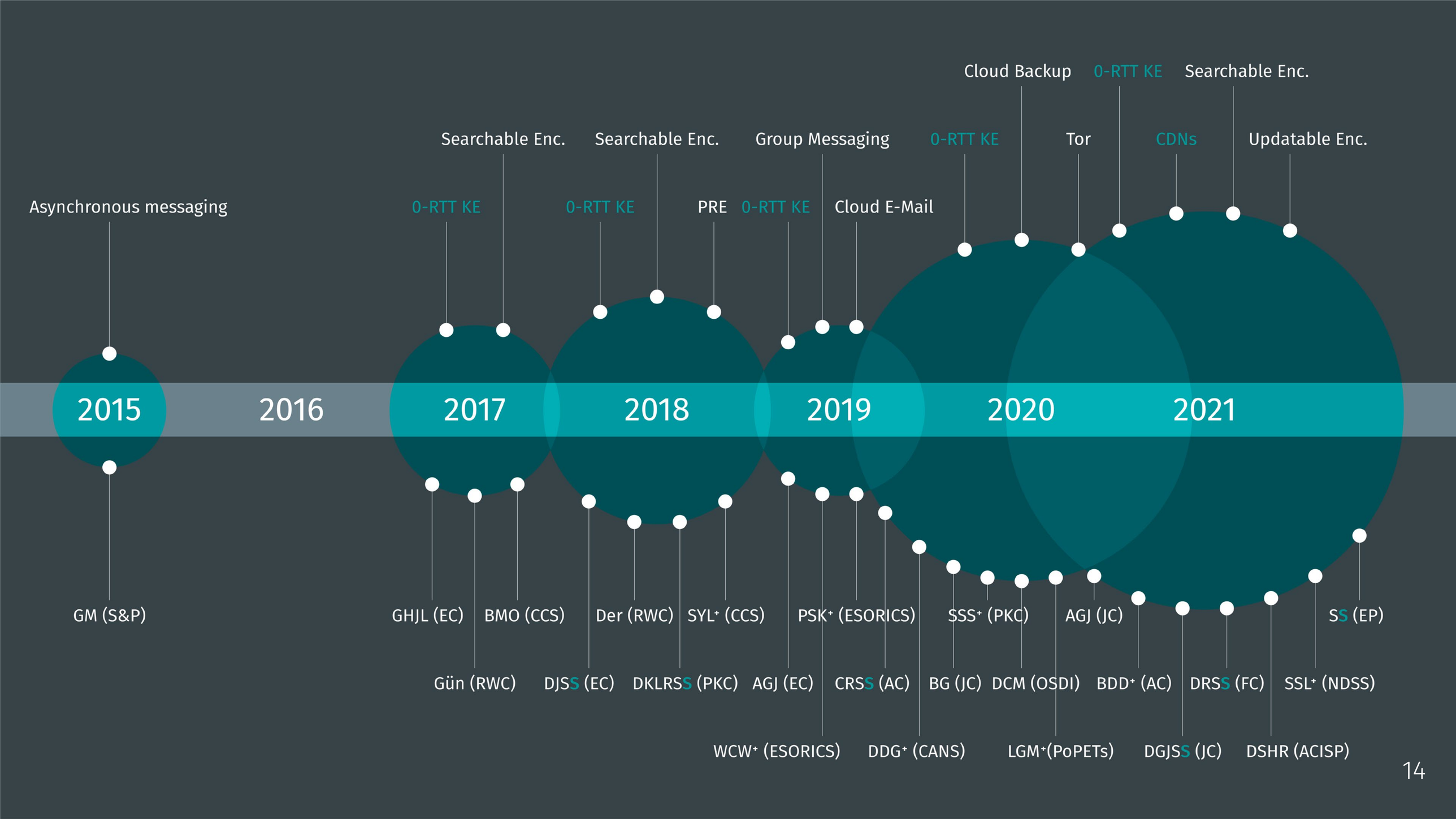
- Problem: loose synchronization required for practicality
- However: no forward-security guarantees for data within epochs
- Solution: Puncturable Encryption (Green & Miers 2015)
- Result: fine-grained forward-security guarantees for *all* data non-interactively

Puncturable Encryption

Tag-based approach (no synchronization required for sufficiently large tag space), fine-grained

Puncturable Encryption

- Common encryption scheme + puncture: $(pk, (sk, \varepsilon)) \leftarrow$ KeyGen, Enc, Dec, Punc
- Encryption: returns $(C, t_1) \leftarrow \mathsf{Enc}(pk, t_1, M)$
- Puncturing: returns $(sk, t_1) \leftarrow \mathsf{Punc}((sk, \varepsilon), t_1)$
- Properties: (sk, t_1) no longer useful to decrypt ciphertexts associated to t_1 (such as (C, t_1)), but still all others with $t_2, \ldots$
- Distinguishing feature: repeated puncturing of secret keys (add more tags to the secret key, exclude more ciphertexts from being decryptable)

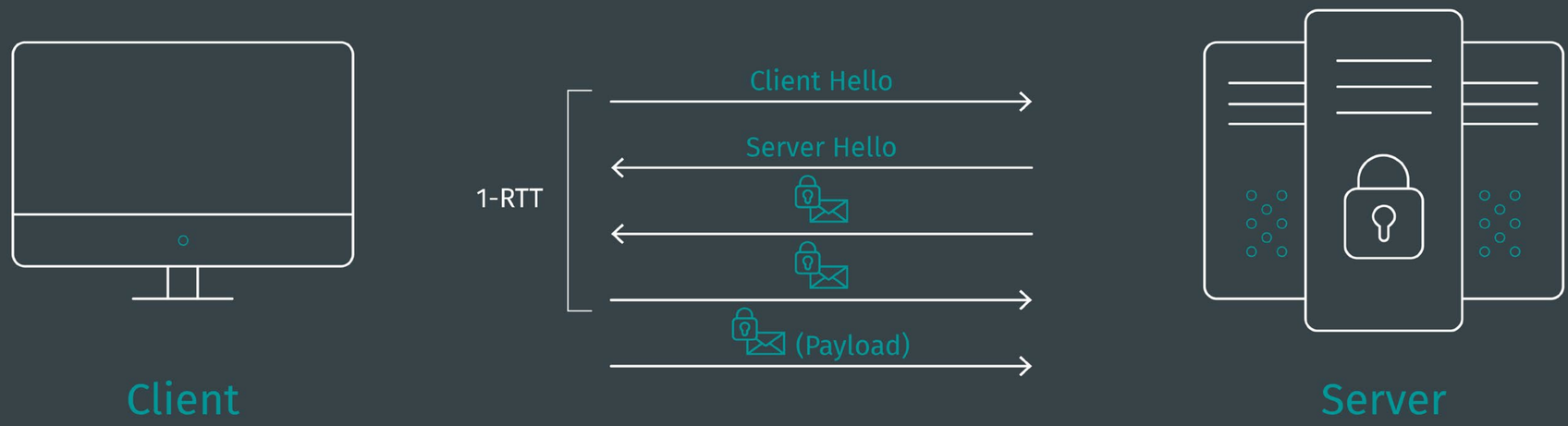

Asynchronous messaging
0-RTT KE
Searchable Enc.
0-RTT KE
Searchable Enc.
PRE
0-RTT KE
Group Messaging
Cloud E-Mail
0-RTT KE
Cloud Backup
Tor
0-RTT KE
CDNs
Searchable Enc.
Updatable Enc.
2015
2016
2017
2018
2019
2020
2021
GM (S&P)
GHJL (EC)
Gün (RWC)
BMO (CCS)
DJSS (EC)
Der (RWC)
DKLRSS (PKC)
SYL+ (CCS)
AGJ (EC)
WCW+ (ESORICS)
PSK+ (ESORICS)
CRSS (AC)
DDG+ (CANS)
SSS+ (PKC)
BG (JC)
DCM (OSDI)
LGM+(PoPETs)
AGJ (JC)
BDD+ (AC)
DGJSS (JC)
DRSS (FC)
DSHR (ACISP)
SSL+ (NDSS)
SS (EP)

Appl. I: 0-RTT Key Exchange with Forward Security and Replay Protection

- Goal: send cryptographically protected payload *non-interactively* (i.e., in 0-RTT) with forward security and replay protection
- Incentive: reduce network communication costs

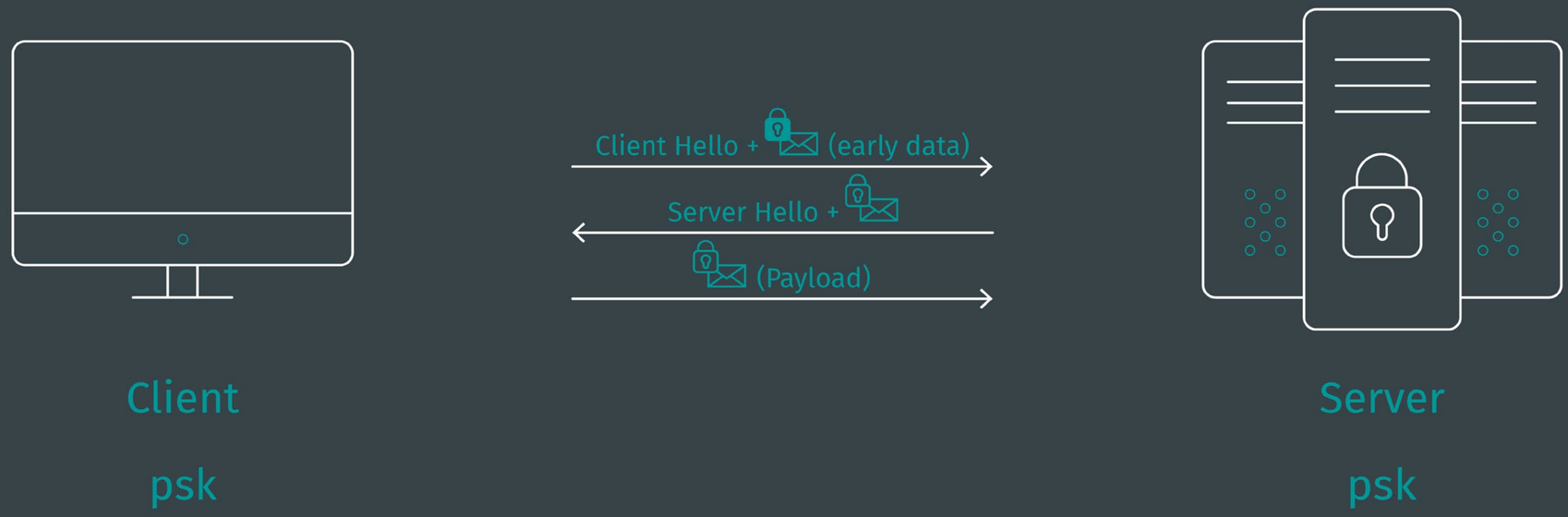
Problem: conventional key establishing modes (e.g., TLS) need at least one round trip (1-RTT) to achieve forward security

Key Establishment with TLS 1.3



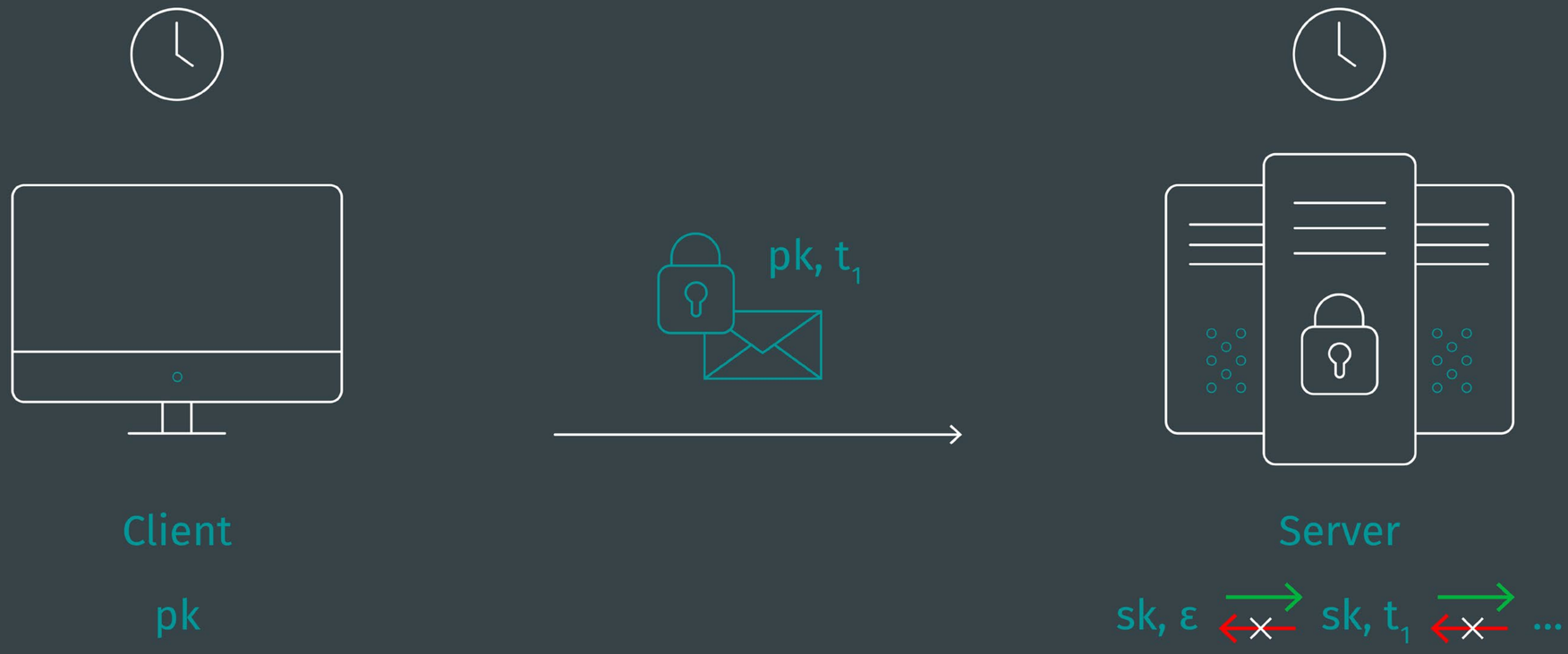
Forward-security guarantees, but 1-RTT before first payload message. Is this necessary?

Key Establishment with TLS 1.3 and Early Data



0-RTT, but no forward-security guarantees for early data.

0-RTT Key Exchange using Puncturable Encryption



Forward security and replay protection in 0-RTT [GHLJ17, DGJSS21],

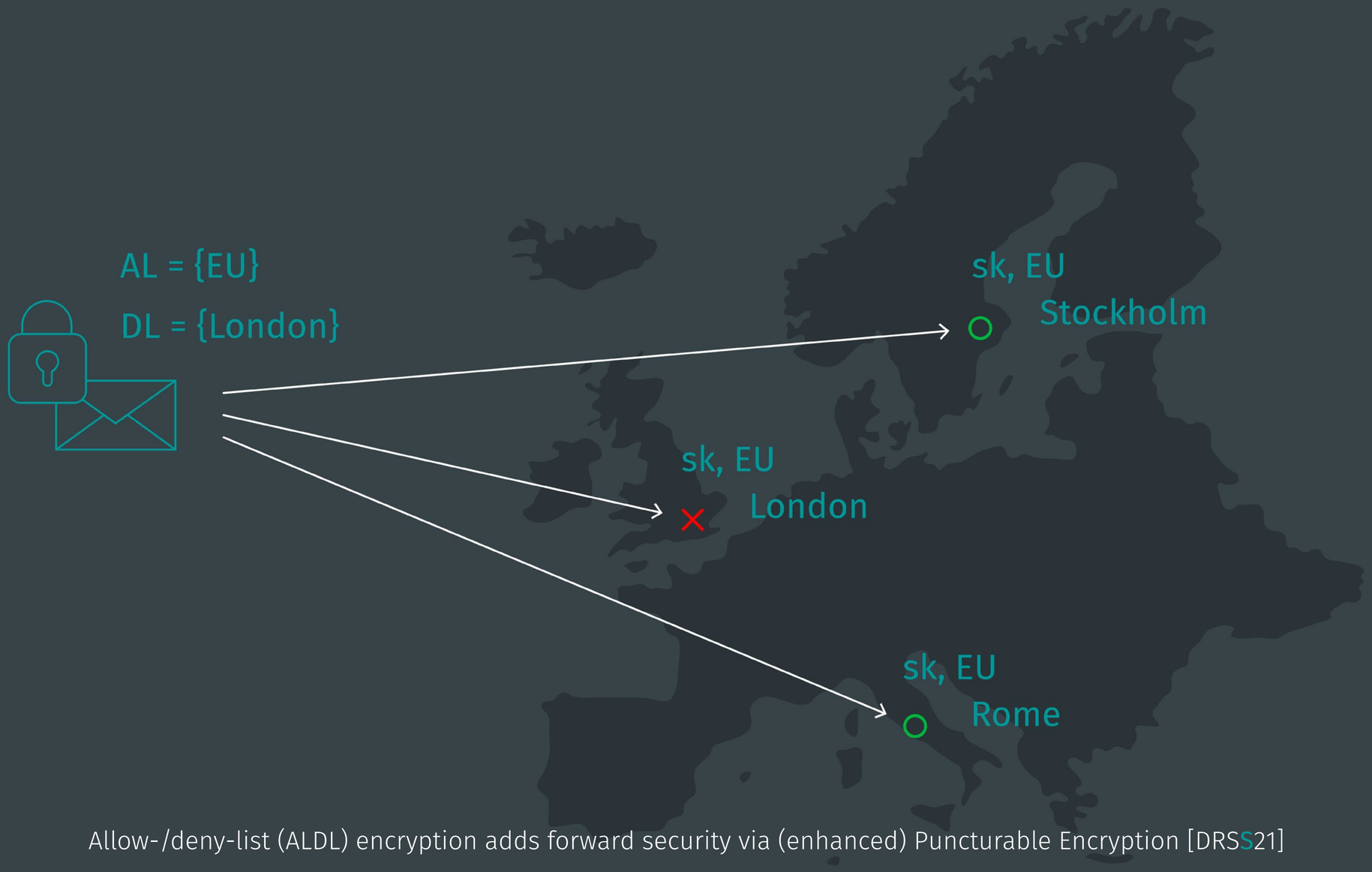
evaluation for QUIC [DDG+20], integration in OpenSSL [Ram21], using time-based puncturable encryption

Appl. II: Forward-Secure Content Distribution Networks

- Goal: Content Distribution Networks distribute TLS secret keys closer to customers
- Incentive: low-latency content distribution
- Solution: restrict access to secret keys – e.g., Cloudflare's Geo Key Manager allows access for certain locations while restricting access for co-locations

Problem: conventional CDNs do not ensure forward security,
i.e., customer keys may leak once a (co-)location key leaks

Forward-Secure CDNs using Puncturable Encryption



Allow-/deny-list (ALDL) encryption adds forward security via (enhanced) Puncturable Encryption [DRSS21]

Takeaways

- High demand in increasing resilience of secret keys
- Puncturable Encryption offers a simple solution on the cryptographic level, particularly for modern distributed and non-interactive scenarios
- Several applications and growing research interest: 0-RTT KE with forward security and replay protection, forward-secure CDNs, Searchable Encryption, mobile Cloud Backup, Tor, Updatable Encryption, ...

AUSTRIAN INSTITUTE OF TECHNOLOGY

Thank you

Christoph Striecks @cstriecks
AIT Austrian Institute of Technology

More on the topic: profet.at/blog/pe

SECREDAS Digital Privacy Safety & Security

COMP4DRONES

FWF Der Wissenschaftsfonds.

netidee FÖRDERUNGEN

KRAKEN

References

[GM] Matthew D. Green, Ian Miers. Forward Secure Asynchronous Messaging from Puncturable Encryption. IEEE Symposium on Security and Privacy 2015.

[GHJL] Felix Günther, Britta Hale, Tibor Jager, Sebastian Lauer. 0-RTT Key Exchange with Full Forward Secrecy. EUROCRYPT 2017.

[Gün] Felix Günther. 0-RTT Key Exchange with Full Forward Secrecy. RWC 2017.

[BMO] Raphaël Bost, Brice Minaud, Olga Ohrimenko. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. CCS 2017.

[DJSS] David Derler, Tibor Jager, Daniel Slamanig, Christoph Striecks. Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange. EUROCRYPT 2018.

[Der] David Derler. Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange. RWC 2018.

[DKLRSS] David Derler, Stephan Krenn, Thomas Lorünser, Sebastian Ramacher, Daniel Slamanig, Christoph Striecks. Revisiting Proxy Re-encryption: Forward Secrecy, Improved Security, and Applications. Public Key Cryptography 2018.

[SYL+] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, Surya Nepal. Practical Backward-Secure Searchable Encryption from Symmetric Puncturable Encryption. CCS 2018.

[AGJ] Nimrod Aviram, Kai Gellert, Tibor Jager. Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT. EUROCRYPT 2019.

[WCW+] Jianghong Wei, Xiaofeng Chen, Jianfeng Wang, Xuexian Hu, Jianfeng Ma. Forward-Secure Puncturable Identity-Based Encryption for Securing Cloud Emails. ESORICS 2019.

[PSK+] Tran Viet Xuan Phuong, Willy Susilo, Jongkil Kim, Guomin Yang, Dongxi Liu. Puncturable Proxy Re-Encryption Supporting to Group Messaging Service. ESORICS 2019.

[CRSS] Valerio Cini, Sebastian Ramacher, Daniel Slamanig, Christoph Striecks. CCA-Secure (Puncturable) KEMs from Encryption with Non-Negligible Decryption Errors. ASIACRYPT 2020.

[DDG+] Fynn Dallmeier, Jan Peter Drees, Kai Gellert, Tobias Handirk, Tibor Jager, Jonas Klauke, Simon Nachtigall, Timo Renzelmann, Rudi Wolf. Forward-Secure 0-RTT Goes Live: Implementation and Performance Analysis in QUIC. CANS 2020.

[BG] Colin Boyd, Kai Gellert. A Modern View on Forward Security. Comput. J. 2021.

[SSS+] Shi-Feng Sun, Amin Sakzad, Ron Steinfeld, Joseph K. Liu, Dawu Gu. Public-Key Puncturable Encryption: Modular and Compact Constructions. Public Key Cryptography 2020.

[DCM] Emma Dauterman, Henry Corrigan-Gibbs, David Mazières. SafetyPin: Encrypted Backups with Human-Memorable Secrets. OSDI 2020.

[LGM+] Sebastian Lauer, Kai Gellert, Robert Merget, Tobias Handirk, Jörg Schwenk. T0RTT: Non-Interactive Immediate Forward-Secret Single-Pass Circuit Construction. PoPETS 2020.

[AGJ] Nimrod Aviram, Kai Gellert, Tibor Jager. Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT. J. Cryptol. 2021.

[BDD+] Colin Boyd, Gareth T. Davies, Bor de Kock, Kai Gellert, Tibor Jager, Lise Millerjord. Symmetric Key Exchange with Full Forward Security and Robust Synchronization. ASIACRYPT 2021.

[DGJSS] David Derler, Kai Gellert, Tibor Jager, Daniel Slamanig, Christoph Striecks. Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange. J. Cryptol. 2021.

[DRSS] David Derler, Sebastian Ramacher, Daniel Slamanig, Christoph Striecks. Fine-Grained Forward Secrecy: Allow-List/Deny-List Encryption and Applications. Financial Cryptography 2021.

[DSHR] Priyanka Dutta, Willy Susilo, Dung Hoang Duong, Partha Sarathi Roy. Puncturable Identity-Based Encryption from Lattices. ACISP 2021.

[SSL+] Shi-Feng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K. Liu, Surya Nepal, Dawu Gu. Practical Non-Interactive Searchable Encryption with Forward and Backward Privacy. NDSS 2021.

[SS] Daniel Slamanig, Christoph Striecks. Puncture 'Em All: Stronger Updatable Encryption with No-Directional Key Updates. IACR Cryptol. ePrint Arch. 2021.

[Ram21] Sebastian Ramacher. https://github.com/ait-crypto/bfe-bf. 2021.