EXPLORING ALGORITHMS

A project report submitted to The University of Manchester
for the degree of Bachelors of Science
in the Faculty of Science and Engineering

2023

Sam Leonard
f41751sl
Supervisor: Professor Bernardo Magri

Department of Computer Science

Declaration

No portion of the work referred to in this project report has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Abstract

Exploring Password Authenticated Key Exchange Algorithms

Sam Leonard, Supervisor: Professor Bernardo Magri

Password Authenticated Key Exchange (PAKE) algorithms are a niche kind of cryptography where parties seek to establish a strong shared key, from a low entropy secret such as a password. This makes the particularly attractive to some domains, such as Industrial Internet of Things (IIOT). However many PAKE algorithms are unsuitable for Internet of Things (IOT) applications, due to their heavy computational requirements. Augmented Composable Password Authenticated Connection Establishment (AuCPace) is a new PAKE protocol which aims to make PAKEs accessible to IIOT by utilising Elliptic Curve Cryptography (ECC), Verifier based PAKEs (V-PAKEs) and a novel augmented approach. This project aims to provide an approachable and developer-focused implementation of AuCPace in Rust and to contribute this implementation back to RustCrypto to promote wider adoption of PAKE algorithms.

Acknowledgements

I would like to thank everyone at absolutely onderful supervisor, RustCrypto, Crypto Hack and my CTF Team (0rganizers). Without your help none of this would have been possible.

Contents

Chapter 1

Context

1.1 Background on PAKEs

1.1.1 What is a PAKE?

PAKEs are interactive, two party cryptographic protocols where each party shares knowledge of a password (a low entropy secret) and seeks to obtain a strong shared key e.g. for use later with a symmetric cipher. Critically an eavesdropper who can listen in two all messages of the key negotiation cannot learn enough information to bruteforce the password. Another way of phrasing this is that brute force attacks on the key must be "online".

1.1.2 A brief history of PAKE algorithms

The first PAKE algorithm was Bellovin and Merritt's EKE scheme[BM92].

1.2 Elliptic Curve Cryptography

1.3 Modern PAKEs

1.4 AuCPace

1.5 Who are RustCrypto?

Chapter 2

Design

2.1 Why Rust?

2.2 Developer Focussed Design

Chapter 3

Implementation

3.1 Overview of RustCrypto and Dalek Cryptography

Chapter 4

Testing

4.1 Creating Test Vectors

Chapter 5

Reflection and Conclusion

5.1 Achievements

5.2 Reflection

5.3 Future Work

Glossary

AuCPace Augmented Composable Password Authenticated Connection Establishment. 4

ECC Elliptic Curve Cryptography. 4

IIOT Industrial Internet of Things. 4

IOT Internet of Things. 4

Online Cryptography Online cryptography is where interactions with the cryptosystem are only possible via real-time interactions with the server. Primarily this is to prevent offline computation. . 7

PAKE Password Authenticated Key Exchange. 4, 7

V-PAKE Verifier based PAKE. 4

Bibliography

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, pages 191–208, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BM92] Steven Michael Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks, May 1992.

[CNPR20] Cas Cremers, Moni Naor, Shahar Paz, and Eyal Ronen. Chip and crisp: Protecting all parties against compromise through identity-binding pakes. Cryptology ePrint Archive, Paper 2020/529, 2020. https://eprint.iacr.org/2020/529.

[GJK21] Yanqi Gu, Stanislaw Jarecki, and Hugo Krawczyk. Khape: Asymmetric pake from key-hiding key exchange. Cryptology ePrint Archive, Paper 2021/873, 2021. https://eprint.iacr.org/2021/873.

[Gre18a] Matthew Green, Oct 2018. https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/.

[Gre18b] Matthew Green. Should you use srp?, 2018. https://blog.cryptographyengineering.com/should-you-use-srp/.

[Ham15] Mike Hamburg. Decaf: Eliminating cofactors through point compression. Cryptology ePrint Archive, Paper 2015/673, 2015. https://eprint.iacr.org/2015/673.

[HL18] Björn Haase and Benoît Labrique. Aucpace: Efficient verifier-based pake protocol tailored for the iiot. Cryptology ePrint Archive, Paper 2018/286, 2018. https://eprint.iacr.org/2018/286.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. Opaque: An asymmetric pake protocol secure against pre-computation attacks. Cryptology ePrint Archive, Paper 2018/163, 2018. https://eprint.iacr.org/2018/163.

[Wu00] Tom Wu. The srp authentication and key exchange system, September 2000. Request for Comments (RFC) 2945.

Appendix A

Python implementation of EKE

While researching Bellovin and Merritt's EKE scheme[BM92], I created a full implementation of the scheme in Python.

```
def negotiate(self):
    # generate random public key Ea
    Ea = RSA.gen()

    # instantiate AES with the password
    P = AES.new(self.password.ljust(16).encode(), AES.MODE_ECB)

    # send a negotiate command
    self.send_json(
        action="negotiate",
        username=self.username,
        enc_pub_key=b64e(P.encrypt(Ea.encode_public_key())),
        modulus=Ea.n
    )

    # receive and decrypt R
    self.recv_json()
    key =
l2b(Ea.decrypt(b2l(P.decrypt(b64d(self.data["enc_secret_key"])))))
    R = AES.new(key, AES.MODE_ECB)

    # send first challenge
    challengeA = randbytes(16)
    self.send_json(challenge_a=b64e(R.encrypt(challengeA)))

    # receive challenge response
    self.recv_json()
    challenge_response =
R.decrypt(b64d(self.data["challenge_response"]))

    assert challenge_response[:16] == challengeA, "Challenge A failed."
```

```
    challengeB = challenge_response[16:]

    # response with challengeB
    self.send_json(challenge_b=b64e(R.encrypt(challengeB)))

    # receive success message
    self.recv_json()
    assert self.data["success"], self.data.get("message", "ChallengeB failed.")

    # store the shared key
    self.R = R
```