Modern Secure Group Messaging with Zero-Trust Servers: the Zpinc Protocol

Zpinc Developers

Abstract

Modern secure messaging consists of several vital features prevalent in messaging protocols such as secure one-to-one and group messaging, resiliency to device compromise and asynchronous messaging. State-of-the-art protocols guarantee an even more extensive set of features. The current server-based solutions assume that the trust in the server itself is inevitable to provide these features.

This paper presents a new *Zpinc* messaging protocol that can provide modern secure group messaging features without trust in the server.

1 Introduction

There are many ways to arrange secure messaging [19]. In this paper, we split messaging services into two main categories: server-based and peer-to-peer services. Server-based services can be split further into centralized or distributed server-based services where messages are forwarded with server nodes on the communication path. Peer-to-peer services exchange messages directly over wired or wireless connections.

The main advantage of a centralized over distributed service is the flexibility and pace of development: a centralized service can be fully upgraded overnight by upgrading the service server-side, assuming the clients already support the new service. For distributed services without centralized control of servers, such upgrades will happen inevitably more gradually. Then again, peer-to-peer services without server infrastructure also enjoy a similar development pace as centralized services.

Server-based services are popular for a reason and may provide features that peer-to-peer services cannot provide well, such as asynchronous messaging. The most popular secure messaging services are server-based (e.g. [12, 13]), centralized or distributed. Also, the new Messaging Layer Security (MLS) protocol is an IETF based recent addition to secure group messaging protocols, and it has been designed to be server-centric [1, 20].

Alas, the Achille's heel of a server-based service is the server itself, as the clients trust it on some level. If an adversary can gain control of the server-side, it causes a significant considerations for the system's overall security (e.g. MLS [1], §7.4). It would be a clear benefit to provide a similar service without such an attack vector.

This paper presents a new messaging protocol called *Zpinc*, which allows building a server-based service with zero-trust in the server itself. Zero-trust implies that the server does not hold any cryptographic material needed for the messaging.

2 Zero-trust server with publish-subscribe pattern

The publish-subscribe pattern is a common way to share data. In brief, publishers of messages can send their message to a service without any subscribers' knowledge, who will receive the messages if they have subscribed to them. A subscriber can also be a publisher. Usually, there may be a need to receive messages from several sources, and a "topic" or a logical "channel" for the messages is provided. In this case, both the publishers and subscribers first register their interest in a particular channel and start sending and receiving messages for the channel.

Popular publish-subscribe pattern services are, e.g. Redis [17] and Message Queuing Telemetry Transport (MQTT) [18]. Even though they are primarily focused on real-time data sharing, MQTT can provide the entire history of published messages or the latest published message for the subscriber.

In the Zpinc protocol, we use publish-subscribe servers to forward messaging data. Some history of messages must be retained to provide asynchronous messaging in Zpinc. Instead of the entire history, the history length can be, e.g. the latest 100 messages.

In the following sections, we refer to sending of publish-subscribe messages with *Broadcast(uid, channel, msg)*, where *uid* is the user identification provided to the publish-subscribe server, *channel* the channel of the messages and *msg* the message delivered to the channel.

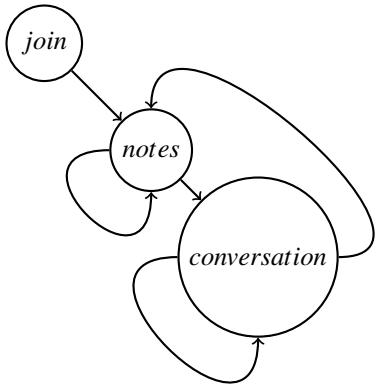

Figure 1: Zpinc protocol state diagram.

3 Zpinc messaging protocol

We build the Zpinc protocol on well-known cryptographic building blocks. We use a memory-hard function (MHF) [14] to create a shared secret key between clients. With the key, the clients can join a correct channel, i.e. gain access to a rendezvous point on a publish-subscribe server. With an incorrect key, the client with bad faith will send its messages into the void of infinite incorrect channels. Every message sent on the channel is encrypted with authenticated encryption with associated data (AEAD). We call these messages *notes*.

After solving the riddle and successfully joining a shared channel, the channel members will initiate a handshake with ephemeral keys. For the handshake phase, we combine composable password authenticated connection establishment (CPace) [2] balanced password-authenticated key agreement with Burmester-Desmedt (BD) [3] conference key distribution system in elliptic curve setting for group messaging. We call the messaging after a successful handshake as a *conversation*. See Figure 1 for a visual view of the communication phases. The cryptographic details of different phases are described in the following subsections.

3.1 Shared key generation with MHF in the join

We use a shared password as the shared secret key in the Zpinc protocol. As passwords have low entropy, they must be passed through an MHF to be secure. If a high entropy shared secret key can be provided by other means, the MHF part could be omitted. As seen in Figure 1, the protocol starts with join processing which is in detail as follows:

$$
\begin{aligned}
pwd &= \text{<password>} \\
key &= MHF\,(pwd) \\
sid_i &\leftarrow \{0,1\}^{k1} \\
(ckey, msgkey) &= KDF\,(key) \\
P_i &= AEAD.enc\,(ckey, uid) \\
rp &= AEAD.enc\,(ckey, channel) \\
enc &= AEAD.enc\,\big(msgkey, sid_i || data_{optional}\big) \\
&Broadcast\,(P_i, rp, enc)
\end{aligned}
$$

After MHF, two independent keys are created with key derivation function (KDF), one for encrypting user and channel information, and another for encrypting messages. The user P_i and channel rp information are constant and sent as part of every message. The join processing is finalized after a message with a random value sid_i is sent to a channel. The message needs not to hold sid only; it could include any data, e.g. a signed user public key, in case user authentication is supported on the channel, or a message.

3.2 Notes messaging with AEAD

After the join part, the actual messages on the channel, referred to as notes, are processed as follows:

$$
\begin{aligned}
P_i &= \text{<from join>} \\
rp &= \text{<from join>} \\
sid_i &= \text{<from join>} \\
msgkey &= \text{<from join>} \\
enc &= AEAD.enc\,(msgkey, sid_i || msg) \\
&Broadcast\,(P_i, rp, enc)
\end{aligned}
$$

Most of the parameters are inherited from the initial join. New messages are concatenated with a random value sid_i, encrypted with msgkey and sent to a channel.

3.3 Zpinc protocol handshake for conversations

If there are more than one member on the channel online, the conversation handshake is initiated. The channel members, referred with $P_{0..n}$, and their sids are learned from initial notes-messages as a first step as follows:

$$
\begin{aligned}
P_i &= \text{<from join>} \\
rp &= \text{<from join>} \\
msgkey &= \text{<from join>} \\
P_{0..n} &= \text{<from rcvd notes>} \\
sid_{0..n} &= \text{<from rcvd notes>} \\
g &= H\left(msgkey, sid_{latest}, P_0||P_1||...||P_{n-1}||P_n\right)^{2} \\
G &= Map2Point\left(g\right) \\
y_i &\leftarrow \{1...m_J\}^{3} \\
Z_i &= y_i \cdot G \\
&Broadcast\left(P_i, rp, Z_i\right) \\
&abort\ if\ received\ Z = y_i \cdot Z_{0..n}\ is\ invalid^{4} \\
X_i &= y_i \cdot \left(Z_{i+1} - Z_{i-1}\right) \\
&Broadcast\left(P_i, rp, X_i\right) \\
&abort\ if\ received\ K = y_i \cdot X_{0..n}\ is\ invalid \\
K &= ny_i Z_{i-1} + \left(n-1\right) X_i + ... + X_{i+n-2} \\
ISK &= H\left(K, sid_{latest}, X_0||X_1||...||X_{n-1}||X_n\right)
\end{aligned}
$$

The latest received sid is used as a parameter to a keyed hash function of which output is passed to Map2Point. Every channel member does this independently with the latest received sid. Map2Point returns a generator used in elliptic curve BD key exchange. Map2Point operation itself "enjoys strong security guarantees" [5, p. 4]. The provided sequence matches CPace specification, extended with BD step for generating and processing X_i. If there are only two members, the BD extension should be omitted.

3.4 Conversation messaging with AEAD

After a new conversation key is generated with a handshake, it is used as the encryption key as follows:

$$
\begin{aligned}
P_i &= \text{<from notes>} \\
rp &= \text{<from notes>} \\
ISK &= \text{<from conversation>} \\
enc &= AEAD.enc\left(ISK, sid_{latest}||msg\right) \\
&Broadcast\left(P_i, rp, enc\right)
\end{aligned}
$$

Only the members who have been part of the handshake can decrypt these messages. This is similar to notes, except the key (ISK) is based on a conversation handshake. If a new sid is received over notes, a new handshake is initiated.

[1] a random value
[2] a keyed hash
[3] a random value between 1 .. (group order - 1)
[4] neutral or out-of-range element

3.5 Zpinc protocol properties

The Zpinc handshake is identical compared to CPace except for the extra BD X_i broadcast and its processing, which can be omitted in the case of one-to-one conversation. The *sid* exchange will reveal the number of conversation parties. The Zpinc handshake will be reinitiated when new parties arrive at the conversation, and leaving parties remain unnoticed until the handshake is reinitiated.

As the modification of the CPace is straightforward BD extension in elliptic curve setting, we expect the presented Zpinc protocol to be secure in case CPace and BD are secure and Diffie–Hellman assumption holds. Both BD and CPace are proven secure [4, 5].

Zpinc conversations should scale reasonably well ($O(n)$) even for large groups as the handshake takes *3n* messages to complete, where *n* is the number of channel members.

4 Available Zpinc protocol features

Zpinc protocol combined with publish-subscribe servers provides a fair amount of guarantees. It provides confidentiality, integrity and asynchronous messaging with notes. The shared secret key and freely selectable user id can provide anonymity and, therefore, deniability. In addition to confidentiality and integrity, conversations can provide session-based forward secrecy (FS) and post-compromise security (PCS). The shared secret key provides group authentication. User authentication can be provided by, e.g. sending a signed public key as part of the initial join message using the trust-on-first-use (TOFU) principle.

The downsides of Zpinc are its lack of FS and PCS when sending notes. Regarding metadata, the encrypted uid and channel information are revealed to an adversary monitoring network traffic without extra transport layer encryption. Even with transport layer encryption, this information can be seen within the zero-trust server as it needs to be delivered to the publish-subscribe entity. The need to exchange the initial secret shared key out-of-band can be considered a downside of Zpinc.

When compared to the state-of-the-art protocols on a high level, the above Zpinc protocol features look fairly comprehensive to categorize Zpinc into modern group messaging class. The most important features can be considered as asynchronous messaging, resiliency to device compromise with FS and PCS and dynamic group memberships [20]. Without trust in servers and its low complexity, it is a fairly different protocol but hopefully a complementing inclusion for zero-trust server use cases.

5 Prototype client implementation

We have created an experimental Zpinc protocol prototype client implementation [6] with the following details:

- MHF: scrypt [7]
- KDF: Blake2 [8] keyed hash with variable input for different keys
- AEAD: XSalsa20-Poly1305 [9, 10]
- Zpinc: Ristretto255 [11] elliptic curve, Blake2 [8] keyed hash with salt

The main objective for the prototype is to show that implementing Zpinc is possible with relatively low effort and that it works as expected. The MHF, KDF and AEAD cryptographic building blocks can be easily changed when needed, even on existing clients, because the publish-subscribe server-side does not depend on the clients' implementation. Also, Zpinc can be implemented with different underlying curve and hash functions; please see CPace specifications [2] for more information.

For improved usability, the prototype client uses a reliable transport to a publish-subscribe server, implements a presence feature with timestamped keepalive messages and can reconnect and resend unsent or lost messages by synchronizing itself into message history in common network glitches.

6 Conclusions

We have presented a new secure group messaging protocol called Zpinc, allowing communication of clients with zero-trust servers. The Zpinc protocol can provide many modern secure group messaging features with low complexity and fair scalability. We consider it a befitting addition among secure group messaging protocols, especially for use cases that cannot count on server-based cryptographic functionality.

Acknowledgments

We want to thank anonymized reviewers for their valuable comments on the early versions of this paper.

Availability

The Zpinc protocol prototype implementation is available at [6].

References

[1] B. Beurdouche, E. Rescorla, E. Omara, S. Inguva, A. Kwon, A. Duric, The Messaging Layer Security (MLS) Architecture, IETF Work In Progress, draft-ietf-mls-architecture-09, August 2022

[2] M. Abdalla, B. Haase, J. Hesse, "CPace, a balanced composable PAKE", IETF Work in Progress, draft-irtf-cfrg-cpace-03, November 2021

[3] M. Burmester and Y. Desmedt, "A Secure and Scalable Group Key Exchange System", Information Processing Letters, Vol 94, Issue 3, pages 137-143, DOI:10.1016/j.ipl.2005.01.003

[4] M. Burmester and Y. Desmedt, A Secure and Efficient Conference Key Distribution System. In A. De Santis, editor, Advances in Cryptology, EUROCRYPT'94, volume 950 of Lecture Notes in Computer Science, pages 275–286. Springer, 1995

[5] Abdalla, M., Haase, B., and J. Hesse, "Security analysis of CPace", https://eprint.iacr.org/2021/114, referenced 11/20/2021

[6] Experimental Zpinc protocol prototype implementation source code repository, https://github.com/ZpincDev, referenced 11/13/2022

[7] C. Percival and S. Josefsson, "The scrypt Password-Based Key Derivation Function", RFC 7914, DOI:10.17487/RFC7914, Aug 2016

[8] M-J. Saarinen, Ed., J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI:10.17487/RFC7693, November 2015

[9] D. Berstein, "Extending the Salsa20 nonce", 2008, https://cr.yp.to/snuffle/xsalsa-20081128.pdf, referenced 10/16/2020

[10] D. Berstein, "The Poly1305-AES Message-Authentication Code." In: Gilbert H., Handschuh H. (eds) Fast Software Encryption. FSE 2005. Lecture Notes in Computer Science, vol 3557. Springer, Berlin, Heidelberg, DOI:10.1007/11502760_3

[11] H. de Valence, J. Grigg, G. Tankersley, F. Valsorda and I. Lovecruft, "The ristretto255 and decaf448 Groups", IETF Work in Progress, draft-irtf-cfrg-ristretto255-decaf448-01, August 2021

[12] Signal Technical Information, https://www.signal.org/docs/, referenced 11/20/2021

[13] Matrix Specification, https://spec.matrix.org/latest, referenced 11/22/2021

[14] C. Percival. Stronger key derivation via sequential memory-hard functions. In BSDCan 2009, 2009.

[15] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, EUROCRYPT'89, volume 434 of LNCS, pages 29–37, Houthalen, Belgium, April 10–13 1990. Springer, Heidelberg, Germany.

[16] Cohn-Gordon, K., Cremers, C.J.F., Garratt, L.: On post-compromise security. In: CSF. pp. 164–178. IEEE Computer Society (2016)

[17] Redis, https://redis.io/

[18] MQTT, https://mqtt.org/

[19] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. 2015. SoK: Secure Messaging. In 2015 IEEE Symposium on Security and Privacy. IEEE, 232–249. https://doi.org/10.1109/SP.2015.22

[20] Matthew Weidner, Martin Kleppmann, Daniel Hugenroth, and Alastair R. Beresford. 2021. Key Agreement for Decentralized Secure Group Messaging with Strong Security Guarantees, Nov. 2021, ACM CCS 2021.