WhatsApp
Key Transparency Overview

Technical white paper

Version 1 published August 22, 2023

Contents

Introduction 3

Terms 4

Device and Key Types 4

Key Transparency 4

System Overview 5

Architecture 5

Client Registration 7

Device removal 8

Primary device removal 8

Lookup Requests 9

Auditing 11

Directory Signatures 12

Conclusion 13

References 14

Final list of required approvals 15

Introduction

This white paper provides a technical explanation of WhatsApp's key transparency system. Please visit WhatsApp's website at www.whatsapp.com/privacy for more information about end-to-end encryption features.

WhatsApp Messenger allows people to exchange messages (including chats, group chats, images, videos, voice messages and files), share status posts, and make WhatsApp calls around the world. WhatsApp messages, voice, and video calls between a sender and receiver that use WhatsApp client software use the Signal protocol outlined in this FAQ article which contains an overview of end-to-end encryption in WhatsApp.

A user can have multiple devices, each with its own set of encryption keys. If the encryption keys of one device are compromised, an attacker cannot use them to decrypt the messages sent to other devices, even for devices registered to the same user. WhatsApp end-to-end encryption means that only the sender and the recipient can read your message and no one in between, not even WhatsApp. These encryption keys participate in a process called key exchange where clients exchange the public portion of their keys through WhatsApp's infrastructure to start an end-to-end encrypted communication session with another user. Verifying the correctness of these encryption keys can be done today via a contact's Encryption page in the application, either by scanning a QR code or verifying a 60-digit number on both the sender and recipient's devices.

In order to have assurance that one user is indeed communicating over a secure channel with another user, manual verification of their security codes must be performed after every key change on a primary device. These key changes occur whenever the application is installed, re-installed, registered, or deregistered to an account. This issue expands quadratically in groups, since each group member has pairwise encryption keys associated with every other member of the group. In a large group especially, it is infeasible to expect anyone to be able to keep up with the necessary frequency of manual verifications to fully assert correctness of all public keys in use in the group.

Fortunately, a cryptographic technique known as key transparency aims to reduce friction by automating much of this process and making it more accessible for users. This document gives an overview of how WhatsApp is providing key transparency from a technical perspective.

Terms

Device and Key Types

- Primary device - A device that is used to register a WhatsApp account with a phone number. Each WhatsApp account is associated with a single primary device. This primary device can be used to link additional companion devices to the account. Supported primary device platforms include Android and iPhone.
- Public Identity Key - A long-term, Curve25519 public key generated at install time.

Key Transparency

- `Hash digest` – The output of a cryptographic hash function. WhatsApp's key transparency implementation uses the Blake3 hash function which emits a 32-byte binary digest value for hash operations.
- `Epoch` – A period of time for which accumulated updates to client keys are published into the directory. This is represented by a monotonically increasing epoch number along with a hash digest which commits to the directory at that point in time.
- `Merkle prefix tree` – A binary tree composed of leaves of information where pairwise hashes are computed to get the value of parent nodes in the tree until an overall root hash is derived at the top of the tree. Leaves are associated with labels that determine their unique position in the tree.
- `Auditable key directory (AKD)` – An open-source, Rust-based implementation of key transparency inspired by the SEEMless protocol with extensions adapted from Parakeet.
- `Publish queue` – A fault-tolerant queueing system for storing public Identity Keys while they're waiting to be published to the auditable key directory.

System Overview

In a typical end-to-end encrypted messaging service, the service provider is responsible for hosting a directory that contains a mapping of contact identifiers (e.g. phone numbers) to public keys. When a user Alice wants to initiate a conversation with another user Bob, Alice issues a lookup request for Bob's public key, and the service provider returns the corresponding entry stored for Bob in this directory.

In a key transparency system, each response to a client-initiated lookup request not only returns the user's public key, but also a cryptographic proof that the returned key has been recorded in the directory. This proof is linked to a public hash digest, which acts as a global commitment to the current state of the directory. Additionally, clients can also query for a history of their own public keys stored in the directory. Responses to these key history requests are also accompanied with cryptographic proofs that link the public keys to a digest for the directory.

There are four primary functions of a key transparency system:

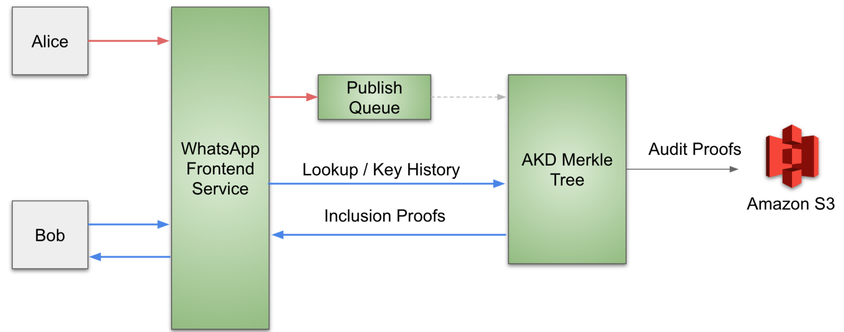
1. Device registration: How new public keys are added to the directory, either by creating new entries or updating existing ones.
2. Lookup requests: How a client can check for the validity of a key it received for another user.
3. Key history requests*: How a client can check for its own history of updates to its keys.
4. Auditing: How the service provider can prove to external auditors that the directory is being maintained in a consistent and append-only manner.

Clients perform two lookup requests when a user attempts to verify their connection with a peer: one lookup for the peer's key, and the other lookup for their own key.

*Note that key history requests are not yet enabled in the current version of WhatsApp's clients. However, we aim to add support for this functionality in the future.

Architecture

Clients connect to the WhatsApp Frontend Service upon new key changes (triggered by client registration) and when opening a contact's info to verify their security code. The frontend service handles these client connections and is responsible for interacting with the other internal services that power key transparency, which include the publish queue and the service which maintains the registry of keys.



The client registration flow (depicted in red) is initiated by a client registration for a new key. This key is added to the publish queue, to be incorporated into a commitment for the next epoch. At a later point in time, a contact of the client can verify (depicted in blue) that the key they received is the most up-to-date version of the key for that contact. This ensures that the client is utilizing the correct version of encryption material for communication between both parties.

WhatsApp also provides auditable proofs of consistency for the transitions between epochs which are published to an Amazon S3 instance which has a public interface for any entity to retrieve the proofs.

Client Registration

At registration time, a WhatsApp client transmits its public Identity Key (along with other material) to the server. The WhatsApp server stores these public keys associated with the user's identifier. Additionally, the key is copied and put into a queue for inclusion into the AKD. The flow of events is as follows:

A client device uploads new public Identity Key to the WhatsApp server

The server stores this key associated with the user's identifier and puts a copy into the publish queue.

Every 5 minutes, a sequencing service empties the queue and generates a changeset of new information to include into the directory. This changeset is summarized with a version number and a new hash digest at the root of the Merkle prefix tree, which make up the new epoch.

Note: The collection and inclusion of public Identity Keys into the AKD is done for all accounts, irrespective of platform.

The information encoded in the leaf of the AKD for an account contains the latest state for an account's public Identity Key changes for the primary device in addition to all of the changes that may have occurred from the previous epoch to the current epoch. This is serialized into a binary format and then hashed as the leaf's digest. The unique position of this leaf is determined by computing a verifiable random function (VRF) on the user's phone number, along with the public key version number and a flag indicating if the key is the latest key for the user or a previous key.

Device removal

In order to handle the deletion of client keys, we employ tombstoning in the auditable key directory (AKD). This is the process of overwriting the raw leaf data with a constant value (i.e. a tombstone). This indicates that the data was once published, but has been removed and cannot no longer be recovered. This is performed on client device removal, deregistration, or after a specified retention time as long as the public Identity Key encoded in the leaf is no longer active.

The hash derived from the original data is still present in the directory; however, its preimage can no longer be validated against. This means that if a client is trying to validate a proof structure with tombstones present, it cannot verify that the original data does indeed result in the encountered hash value. Client verification proofs trust that the hash is accurate without verifying it from a raw value. However, tombstones are verified to be contiguous, meaning that once a tombstone is encountered, all older entries must also be tombstoned. This protects from tombstoning newer information without cleaning the entire account history past a given point.

Primary device removal

When the primary device is removed, it triggers a total deletion of the account. Because the AKD is immutable in history, this process tombstones all entries for the given account (including the most recent value).

The process of a primary device removal is the following:

1. The primary device requests account deletion with the server
2. The server schedules deletion of all properties related to the account
3. All versions of the given account state are tombstoned in the directory

Lookup Requests

A lookup request asks the server to generate a proof for the latest public Identity Key for a given identity in the directory. Checking the public Identity Key(s) for another client device against a root hash asserts that the value the server has provided on the other device's behalf is a part of the directory committed by the specified root hash. This helps to prevent server equivocation: namely, the scenario in which the server attempts to provide an inconsistent value (across requesting clients) for an account at a specific epoch.

The process of generating and verifying a lookup proof is as follows:

1. A client opens either the Contact Info or Encryption page for a given chat participant
2. The client sends a request to the server for a lookup proof for the contact they've opened with the contact's identifier (phone number) and the public key they have for that contact, along with their own identifier and Identity Key public key, which allows as a self-verification of their latest key.
3. The server checks:
    a. First, if the intended contact's key is "pending" and updated in the change queue, then returns a PENDING code to the client
    b. Second, if not pending, the server queries the AKD to generate the requested lookup proof providing it to the client.
        i. Should the expected key not be found in the AKD or is a different value, then the server returns NOT_FOUND indicating that the client won't be able to verify the result automatically and should resort to manual verification.
4. If successful, the proof is returned to the client, which deserializes and computes the necessary hash operations to validate the proof.
    a. Additionally, the client verifies that the public Identity Key they received from the server initially matches the contents of the Lookup Proof so they can verify they are validating the correct public Identity Key value.
    b. Lastly the client verifies that the signature on the locally derived root hash (see below) is valid and signed with the public key included in the WhatsApp application.

5. The client shows a visual indication of the status of the verification process on the Encryption page. The possible outcomes are:

    a. Successful verification: The client device verified that the public Identity Key that was given initially by the server matches what is included in the AKD for both the contact and their own device.

    b. Pending verification: If the target account or their own account have changed a public Identity Key very recently, it is possible that they are pending inclusion into the AKD, and are presently enqueued. Therefore the client should try verification again later. This state does **NOT** indicate anything is wrong with the session.

    c. Failed verification: In the event that (1) enough time has elapsed and either (2a) the public Identity Key provided in the proof does not match the public Identity Key provided by the server or (2b) the target account or their own account is unable to be found, the client device assumes the end-to-end encryption session cannot be verified automatically and should be verified manually with the contact in question.

These above steps are expected to complete within a short time frame in most cases (on the order of seconds).

Auditing

In addition to the security guarantees that the combination of peer and self lookup proofs give, verifying the continued correctness of the entire network is done through audit proofs. Audit proofs, also referred to as append-only proofs or consistency proofs, are cryptographic proofs which provide evidence to the append-only nature of the changes to the directory over time. They assert that at no point in time has the directory been reconstructed with past history deleted or modified in an irrecoverable way.

Audit proofs are made available to the public so that anyone can verify them and perform the role of public auditor, organizations and individuals alike. They are available here and a local auditing solution for parsing the audit proofs and verifying them is provided in our open-source AKD library.

The process for audit proof generation and verification is as follows:

1. After the sequencing process handles the changes for the next epoch of time to the AKD storage layer, we also generate audit proof for the time period epoch T to T+1 (the next epoch of the directory).

2. This proof is serialized in a backwards-compatible format (protobuf) and uploaded to an AWS S3 bucket for public consumption.

    a. The S3 bucket has enabled the WORM (write-once-read-many) model with a 5-year retention period. This helps to ensure that once an object is written to the storage layer, it cannot be deleted or updated for at least 5 years.

    b. Public access is provided through a public web portal which AWS maintains and includes fair-use limitations such as DDOS protections to prevent denial of access to the audit proofs.

3. Any public entity can utilize our open-source library to connect to the public endpoint and verify audit proofs for any epoch change in the auditable key directory. The changes emit a hash digest for the latest epoch in the audit proof, which can be matched against the root hash derived by clients.

Directory Signatures

After generating an audit proof of the changes and publishing it to the public repository, the sequencing process additionally generates a signature on the root hash of the directory with a private key only accessible by that process. This signature hardens the AKD against other entities from generating changes based on the current state of the directory, commonly referred to as a split-view attack, where the server or attacking entity provides a divergent view of the key directory via falsified proof structures.

This signature asserts that the legitimate WhatsApp sequencing process generated the changes which are being client-side validated in the form of a lookup proof (and in the future, key history proof), without additional communication between client and server. The public key for this signing process is included in the distributed WhatsApp client binary applications.

Conclusion

Key transparency solutions can be deployed by end-to-end encrypted messaging applications to allow for users to more easily validate the authenticity of the keys they receive from their service provider. The proofs of consistency for key updates act as a supplement, but not a complete replacement, to the existing manual verification of key fingerprints which are provided on the Encryption page for a contact.

Our goal with this feature is to provide a system in which verification of user keys can occur more automatically, especially for users who do not verify their security codes manually. WhatsApp is committed to building technologies that safeguard the content of users' messages, which includes launching end-to-end encryption at scale in 2016, enabling end-to-end encrypted backups in 2021, and providing key transparency for automatic key verification moving forward.

References

1. WhatsApp Encryption Overview - technical white paper
   https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf

2. About end-to-end encryption - Help center article
   https://faq.whatsapp.com/820124435853543

3. About security-code change notifications - Help center article
   https://faq.whatsapp.com/1524220618005378

4. Auditable key directory (AKD) implementation
   https://github.com/facebook/akd

5. SEEMless: Secure End-to-End Encrypted Messaging with less trust
   https://eprint.iacr.org/2018/607

6. Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging
   https://eprint.iacr.org/2023/81