Review for : Automatic Detection of Fake Key Attacks in Secure Messaging

Siddhant Gupta[1], Samyak Jain[2] and Advait Shrivastava[3]

[1]2022201037, [2]2022201048 and [3]2022201069

March 31, 2023

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

1 Introduction

Today many popular instant messaging applications, such as WhatsApp and Signal, provide end-to-end encryption for billions of users. They rely on an application-specific, centralized server to distribute public keys and relay encrypted messages between the users. Therefore, they prevent passive attacks but are vulnerable to some active attacks. A malicious or hacked server can distribute fake keys to users to perform man-in-the-middle or impersonation attacks. To counter fake key attacks, most chat applications provide users with a method to verify each others' public keys. This verification is done by manually comparing a key fingerprint or scanning a QR code of the fingerprint. Research and studies have found that users are generally oblivious to the need to verify public keys and are unlikely to authenticate, leaving them vulnerable to an attack.

The paper aims to relieve users of this burden entirely. In this paper, three novel approaches are discussed that leverage the ideas of client auditing and anonymity to help detect fake key attacks automatically. The third approach is a combination of the first two approaches. The approaches are:

1. Key Transparency with Client Auditors (KTCA)
2. Anonymous Key Monitoring (AKM)
3. Key Transparency with Anonymous Client Auditors (KTACA).

2 Adversary Model

Adversary A is an attacker that has control over the server, and hence it can access all encrypted messages and metadata from the communication between clients and server and has control over public key distribution. Description of Attacks: When users request the public key, the attacker creates a fake public key and distributes it to them. This attacker can perform two kinds of attacks.

1. Man In The Middle Attack (MITM)
2. Impersonation attack

3 APPROACHES

3.1 KEY TRANSPARENCY WITH CLIENT AUDITORS

KTCA is an adaptation of key transparency that relies on secure messaging clients to audit the server. The server maintains a Merkle binary prefix tree of all the registered client's public keys, inspired by CONIKS [3]. Merkle tree provides an efficient method for clients to verify that their key is included in the current tree without obtaining a full copy of the tree.

At each epoch, the server generates a Signed Tree Root (STR) by signing the root of the Merkle tree, along with other metadata, such as a hash of the previous epoch's STR and epoch number. Along with current STR, the server also provides a client with a Proof of Inclusion (PoI) for their key. PoI contains the leaf node corresponding to the client's key and the hash of each sibling sub-tree along the path from the leaf node to the root of the Merkle tree. To verify the PoI, the client begins at the leaf, hashes the hash of the leaf along with the hash of the leaf of the sibling node to produce the value of the parent node.

All clients perform the following for key monitoring and server auditing : (1) monitor their own key, (2) monitor the keys for all their contacts, (3) audit the server to detect equivocation, and (4) perform short-lived attack monitoring.

1. For monitoring their own key, clients verify (a) that the server is publishing a linear history of STRs by confirming the previous STR's hash is in the current STR, (b) that the STR's signature is valid, and (c) that their public key is in the tree using PoI. If a client doesn't receive STR or PoI within 2δ time after the starting of an epoch, or the STR is invalid, it considers this an attack, and disconnects from the server.

2. For client Bob to verify the client Alice's key is in the current tree, server gives client Bob the key for client Alice that is included in *STRe* and its PoI.

3. For detecting server equivocation, client sends *STRe* to their contacts during each epoch, immediately upon receiving it. If two conflicting STRs are found, the client sends Proof of Misbehavior (PoM) to all its contacts and they relay the same to their contacts.

4. Client performs short-lived attack monitoring to detect attacks whether A quickly restores a correct key, whenever key update is received.

3.2 ANONYMOUS KEY MONITORING

To overcome KTCA's vulnerability to graph partitioning attacks, the paper presents anonymous key monitoring. Server distributes public keys to clients directly and also supports an Anonymous Key Request (AKR) for key monitoring. An AKR is an unauthenticated public key request sent through an anonymous network to retrieve a key from the server.

Client performs the following to detect fake key attacks :

1. Client monitors its key using AKR at the start of each epoch to make sure server consistently distributes its key.

2. Whenever key update of client contact's is received, client monitors contact's key using an AKR at the beginning of each epoch from epoch $e + 1$ to $e + m$ where m is the number of monitoring requests.

3. Similar to KTCA whenever a key update is received, the client performs short-lived attack monitoring to detect whether A quickly restores a correct key.

3.3 KEY TRANSPARENCY WITH ANONYMOUS CLIENT AUDITORS

KTACA combines key transparency and anonymity and overcome shortcomings of both the above methods, and to achieve the best of both approaches. Similar to KTCA, the server maintains a Merkle tree containing the public key of all registered clients and it supports an Anonymous STR Request (ASR) which is an unauthenticated STR request sent through an anonymous network to retrieve the STR for an epoch from the server. Using this ASR, Clients audit for equivocation by instead of exchanging STRs with their contacts as in KTCA.

1. All clients retrieve an STR and Proof of Inclusion (PoI) at the start of every epoch. Clients verify (a) the server is publishing a linear history of STRs by checking

previous STR's hash is present in the current STR, (b) the STR's signature is valid, and (c) that their public key is in the tree using PoI. If STR is invalid or if client does not receive an STR or PoI within $2 \cdot \Delta$ time it consider server under attack and disconnects

2. Clients retrieve ASR at the start of every epoch and if a client has conflicting STRs and detects an attack with a Proof of Misbehavior.

3. Client receives a POI for verifying the key is in the tree for every key update and new key lookup.

4. Similar to KTCA and AKM, clients perform short-lived attack monitoring to verify if an attacker quickly restores a correct key on every key update.

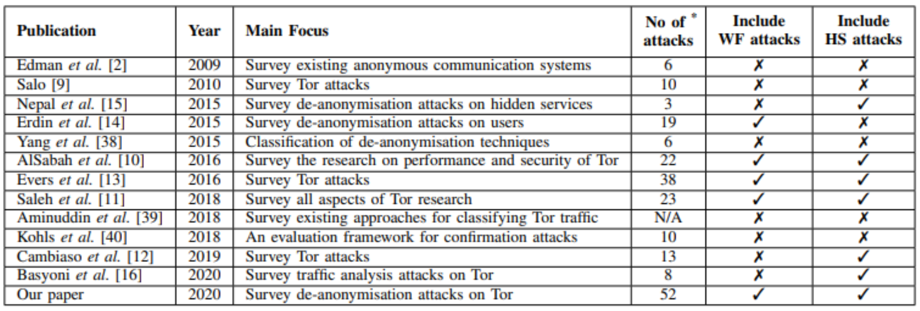
4 SHORT-LIVED ATTACK MONITORING

Short lived attack monitors for the attack where fake key is distributed and original key is restored before the epoch ends so that the fake key does not get identified by client in e+1. This can be prevented in KTCA and KTACA, by allowing at most one key change per epoch per epoch to ensure that any new key will be in e+1. Also, each client maintains a key update history for its contacts, then checks for duplicates when a key update is receieved.

5 CRITIQUES

5.1 Based on Assumptions

1. Anonymous networks are used in AKM for Anonymous Key Request (AKR) which basically send unauthenticated public key requests to retrieve a key from the server anonymously, and in KTACA for Anonymous STR Request (ASR) which is an unauthenticated STR to retrieve STRs anonymously. In the paper anonymous networks like Tor are suggested to be used as it is assumed to be one of the best anonymous network provider in the paper. However, over the years there have been various de-anonymisation attacks carried out by different organizations including governments and law enforcement agencies. Some of known published attacks have been listed below taken from the paper *De-anonymisation attacks on Tor: A Survey* [1] [2]



Publication	Year	Main Focus	No of * attacks	Include WF attacks	Include HS attacks
Edman *et al.* [2]	2009	Survey existing anonymous communication systems	6	✗	✗
Salo [9]	2010	Survey Tor attacks	10	✗	✗
Nepal *et al.* [15]	2015	Survey de-anonymisation attacks on hidden services	3	✗	✓
Erdin *et al.* [14]	2015	Survey de-anonymisation attacks on users	19	✓	✗
Yang *et al.* [38]	2015	Classification of de-anonymisation techniques	6	✗	✗
AlSabah *et al.* [10]	2016	Survey the research on performance and security of Tor	22	✓	✓
Evers *et al.* [13]	2016	Survey Tor attacks	38	✓	✓
Saleh *et al.* [11]	2018	Survey all aspects of Tor research	23	✓	✓
Aminuddin *et al.* [39]	2018	Survey existing approaches for classifying Tor traffic	N/A	✗	✗
Kohls *et al.* [40]	2018	An evaluation framework for confirmation attacks	10	✗	✗
Cambiaso *et al.* [12]	2019	Survey Tor attacks	13	✗	✓
Basyoni *et al.* [16]	2020	Survey traffic analysis attacks on Tor	8	✗	✓
Our paper	2020	Survey de-anonymisation attacks on Tor	52	✓	✓

WF - Website Fingerprinting, HS - Hidden Service, N/A - Not Applicable
* This number denotes the number of de-anonymisation attacks on Tor referenced in the paper.

It is important to note that these attacks are taken from papers published on high-quality venues or papers with at least 20 citations. However the aim of some of the attacks is to find out who is visiting a particular website or finding out what websites are being visited by a targeted user but that reduces the vulnerabilities possessed by the Tor network.

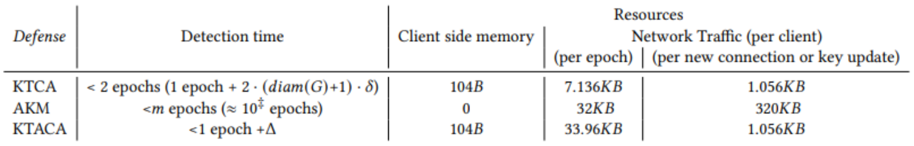
2. Certain flaws in anonymous key monitoring :-
   (a) Legal issues: In many countries, monitoring through anonymous key logging is illegal unless it is authorised by the law enforcement organizations. [4]
   (b) Paper assumes that in AKM method only short messages are sent using anonymous key monitoring but this might fail when the length of message is large.

5.2 Based on Technical Approaches

1. Authors suggested using the Merkle tree to store the hashed values of keys of all the clients. Though the Merkle tree is a useful data structure, there are a few disadvantages to using Merkle tree for key verification:-
   (a) Verification using the Merkle tree depends on the hashed value of the root/ Single root hash to verify the whole tree causes a single point of failure.
   (b) When the size of the dataset is less, the Merkle tree works very well but as the size increases, the overhead of constructing a tree also increases which might degrade the performance.
   (c) When the number of users are large, building and verifying can be difficult because it might require a lot of CPU usage for computation of hash values.
   (d) The assumption taken by authors while using the Merkle tree is that the hash function is collision-free. If this assumption is violated, there is a chance that the Merkle tree might be compromised.
2. In KTCA method, “If a client does not receive an STR within $2 \cdot \delta$ time after the beginning of an epoch it considers this an attack, and the client disconnects from the server”, similarly in KTACA method “If a client Alice does not receive a response for its AKR within $2 \cdot \Delta$ where Δ is the maximum delay in the anonymous network between a client and the server, the client assumes the server is avoiding detection and considers it an attack.” This can lead to false alerts as congestion in network traffic can delay the response time from $2 \cdot \delta$ (KTCA) or $2 \cdot \Delta$ (in the case of KTACA) and clients may interpret the server under attack.

5.3 Based on Analysis

1. In KTCA and KTACA approaches, when the clients join back after a long disconnection, it retrieves all the missed STRs and POIs. It validates the linear history of STR by confirming the previous STR’s hash is the next STR and validates all the POIs. This can be a lot of computation required from the client side and cause excess traffic in the network. The below table represents detection time, client-side memory, and network traffic for one epoch for all three approaches.



Defense	Detection time	Client side memory	Resources	
			Network Traffic (per client)	
			(per epoch)	(per new connection or key update)
KTCA	< 2 epochs (1 epoch + $2 \cdot (diam(G)+1) \cdot \delta$)	$104B$	$7.136KB$	$1.056KB$
AKM	<m epochs ($\approx 10^{‡}$ epochs)	0	$32KB$	$320KB$
KTACA	<1 epoch +Δ	$104B$	$33.96KB$	$1.056KB$

It’s worth noting that network traffic is around 34KB for one epoch in the KTACA approach. Now if the client is connecting after a long gap, then there will be a huge increase in the network traffic and client-side computations. [6]

2. According to the KTCA approach, if Alice receives an incorrect STR, she considers this as an attack because the attacker might have replaced the original key of some client(let's say bob) with a fake key. In this scenario, Alice will not be able to discover that Bob's key is fake; she just knows that a key is compromised in the network. Only the owner client of the key can confirm if its original key is present in the STR or not.

3. Some attacks discussed in the paper are detectable without any PoM (proof of misbehavior). So, a dishonest client could falsely accuse a server of a fake key attack. Likewise, an adversary can respond to an accusation by accusing the client of making a false report. There is no way for a third party to determine which claim is correct.

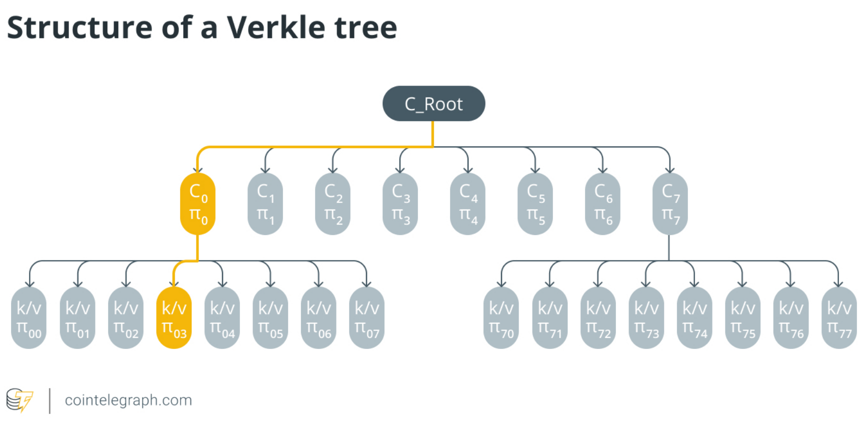
6 IMPROVEMENTS

1. **Verkle Trees**

   Verkle trees are similar to Merkle trees and allow you to organize a considerable quantity of data and create a brief "witness" of each item of data or group of related pieces that can be confirmed by someone who has access to the tree's root.

   However, the Verkle tree's most important feature is its proof-size efficiency. A Verkle tree requires less than 150 bytes to produce a proof for a tree with a billion data points, compared to a typical binary Merkle tree's around 1 kilobyte. Verkle trees utilize a proving system called Polynomial Commitments, relying upon polynomial functions to describe data.

   The advantage of using Verkle Trees for key verification is that they allow for efficient storage and verification of large sets of keys. Verkle Trees are designed to minimize the amount of data that needs to be stored and processed while still providing high cryptographic security guarantees. This makes them well-suited for applications with limited storage and computation resources, such as decentralized networks or embedded devices. [5]



2. **Strong Consistency Required**

   Chat/messaging applications that are popular and installed by millions of users require multiple servers for load balancing and scalability. In such cases, these servers must maintain a consistent copy of STRs to prevent false detection of fake key attacks. This can be achieved by implementing strong consistency protocols. We can dedicate one server that generates STR during the start of each epoch and distribute it to all the other servers.

3. **Implementing Out of band channel for client-to-client communication**

   In the case of client-target impersonation attacks in the three approaches(KTCA, AKM, KTACA), only the targeted clients can detect the attack. They have to then manually alert all their contacts about the attack. We can design an out-of-band channel for client-to-client communication to relay the attack information among clients and their connections effectively.

4. **Alternatives to digital signatures in Merkle Tree**

   KTCA and KTACA use digital signatures for signing the root of the Merkle tree, and this signed data gets forwarded to clients for key verification. While digital signatures are widely used in Merkle trees, there are some alternatives that offer certain advantages in some specific use cases. They are:

   - Threshold Signatures: They offer improved security and availability properties over traditional digital signatures, and they have been proposed for use in Merkle Trees. In threshold signatures multiple parties can jointly sign a message such that any subset of the parties can produce a valid signature. In a chat application that uses Merkle trees for key verification, each user's public key is included in the Merkle tree, and the root of the tree is signed by a set of trusted parties using threshold signatures. When a user wants to verify the authenticity of another user's public key, they can request the signed Merkle tree root from the trusted parties and compare it to the root of the Merkle tree they have received. [7]
   - BLS signatures: They offer certain advantages over traditional digital signatures such as smaller signature sizes and more efficient verification. Many researchers have proposed the usage of BLS signatures over traditional digital signatures in Merkle trees. In chat applications that use Merkle trees for verification of keys, every client's public key is included in the Merkle tree, and the root of the tree is signed by a trusted party using BLS signatures. [8]

5. **Whitelisting**

   We can prevent chat applications from being prone to fake key attacks by using the whitelisting technique. Here we will assign clients public keys from a set of trusted keys and only allow messages to be encrypted using these. However, one of its drawbacks is that if the application scales and has millions of users, the set might get exhausted.

6. **Alternatives to TOR Network** The paper implements the TOR network for anonymous communication in AKM and KTACA. While TOR is widely popular for providing anonymity to clients, it also has some disadvantages, such as slow

speed, vulnerability to timing attacks, and susceptibility to Sybil attacks. So, for chat applications, some alternatives are more suitable than TOR, such as:

- I2P Anonymous Network can be used to develop a decentralized chat application. It has privacy protection and gives strong anonymity with end-to-end encryption.
- P2P Network can also be used for developing a decentralized messaging application that uses public keys for encryption and authentication. P2P networks do not rely on central servers, making it difficult for an adversary to intercept and surveil messages.

7. **Use Private Information Retrieval (PIR) technique in anonymous networks**

   In AKM, every client sends a distinct, single-key AKR(anonymous key request) for each key they monitor at the start of every epoch. This approach helps in anonymisation from adversary by sending bulk requests. An alternate and better method to prevent de-anonymisation while sending bulk requests is to use the PIR technique. Through the PIR technique, a client can retrieve information from a database in possession of a server without revealing which item it retrieved. AKR by clients can leverage this technique to send bulk requests without compromising anonymity. PIR hides which keys clients request, and bulk requests are not visible to the server.

References

[1] De-anonymisation attacks on Tor: A Survey" published in IEEE Communications Surveys Tutorials in 2021 edition by I Karunanayake, N Ahmed, R Malaney, R Islam, SK Jha.

[2] "Breaking Tor's Anonymity by Modifying Cell's Command" published in 2022 IEEE Symposium on Computers and Communications (ISCC) by Yi Qin, Jiahe Wu, Futai Zou, Yue Wu.

[3] "Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. 2015. CONIKS: Bringing Key Transparency to End Users. In USENIX Security Symposium

[4] https://vpnoverview.com/privacy/anonymous-browsing/is-tor-legal/

[5] https://cointelegraph.com/explained/merkle-trees-vs-verkle-trees-explained

[6] Section 9.2 in https://dl.acm.org/doi/pdf/10.1145/3548606.3560588

[7] https://bitcoinops.org/en/topics/threshold-signature/

[8] https://crypto.stanford.edu/ dabo/pubs/papers/BLSmultisig.html