# On the Security of E2EE Device Linking Protocols: QR Sniffing Attacks and SAS Authentication Defense

Md Shahidur Rahaman
Texas A&M University
College Station, Texas, USA
mdshahidur_rahaman@tamu.edu

Nitesh Saxena
Texas A&M University
College Station, Texas, USA
nsaxena@tamu.edu

## Abstract

Widely adopted end-to-end encrypted (E2EE) messaging apps, such as WhatsApp, WeChat, and Telegram, allow the user to link a secondary device (e.g., laptop/desktop) to the primary device (mobile phone), syncing the chats between the two devices and enabling access across devices. The device linking process is critical because if the attacker can compromise its security, it compromises the E2EE security of chats/calls between the user and all their contacts. This paper shows that current device-linking protocols have a fundamental problem that makes them vulnerable to eavesdropping attacks and proposes a robust device-linking protocol based on the notion of Short Authentication Strings (SAS) to mitigate this threat.

Our attack stems from currently deployed device linking protocols directly transferring secret cryptographic material, using session keys that are derived by embedding it into a QR code displayed on the secondary device captured via the primary device. We demonstrate two attack scenarios where the attacker can grab this QR code to compromise device linking. The first attack scenario, `QRSniffer-Browser`, exploits the secondary device registration process in the browser, where malicious code can execute without the user's knowledge, enabling an attacker to mirror the attacker's account and potentially inject malicious messages. The second scenario, `QRSniffer-Prox`, targets users attempting to mirror their account on a desktop/laptop application, where an attacker within proximity can eavesdrop on the QR code and gain unauthorized access. To counter these vulnerabilities, we propose a SAS-based authentication mechanism, `SASLinker` that requires users to validate a short protocol fingerprint string on both the primary and secondary devices during registration. Importantly, even if an attacker captures the SAS by eavesdropping it, it would not help the attacker as the SAS does not contain secret key material, only material for authenticating keys. Therefore, breaking the scheme is infeasible by simply eavesdropping the SAS, unlike current device linking protocols. We show that SAS validation can be performed by having the user copy the alphanumeric SAS code displayed on the secondary device over to the primary device or by capturing the QR representation of the SAS code by taking a picture. By implementing this device-linking solution, messaging services can effectively protect their users from unauthorized access and ensure the integrity of their communications.

## CCS Concepts

• **Security and privacy** → *Social network security and privacy.*

## Keywords

End-to-End Encryption (E2EE), Device Linking Security, QR Code Vulnerabilities, Short Authentication String (SAS), WhatsApp Security

**ACM Reference Format:**
Md Shahidur Rahaman and Nitesh Saxena. 2018. On the Security of E2EE Device Linking Protocols: QR Sniffing Attacks and SAS Authentication Defense. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (WiSec 2025).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

Communication technology's evolution has dramatically transformed how people interact and stay connected. End-to-end encrypted (E2EE) messaging applications have emerged as a cornerstone of this transformation, offering real-time communication capabilities that transcend geographical boundaries [25] [17]. These platforms have facilitated personal and professional interactions and have become integral to the daily lives of billions of users worldwide [19]. E2EE messaging application's convenience, speed, and multimedia capabilities have made them indispensable tools in the modern digital era [23] [2]. WhatsApp, WeChat, and Telegram stand out among the myriad applications available today due to their extensive user bases and feature-rich platforms. WhatsApp alone boasts over 2 billion users globally, while WeChat and Telegram have approximately 1.2 billion and 500 million users, respectively [24] [35]. These applications provide a wide range of functionalities, from text and voice messaging to video calls and file sharing, making them versatile tools for communication [22] [33] [32].

The widespread adoption of these services underscores their profound impact on how people connect and share information. However, the rapid proliferation of end-to-end(E2EE) communication has also brought significant security concerns. Users entrust these platforms with sensitive personal information, from private conversations to multimedia files and contact lists [36]. The security and privacy of user data are paramount, as breaches can lead to identity theft, unauthorized access to private information, and other malicious activities [15]. Consequently, safeguarding user data and ensuring secure communication channels are vital for maintaining user trust and the integrity of these platforms. Security


*WiSec 2025, June 30– July 03, 2025, Arlington, Virginia*

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

vulnerabilities in E applications are not merely theoretical; they have tangible implications for millions of users worldwide.

For instance, in 2019, WhatsApp disclosed a vulnerability that allowed spyware to be installed on users' devices via a simple missed call [8] [31]. These incidents highlight the pressing need for robust security measures to protect users from threats. One of the critical areas of concern is the linking device mechanism and authentication method, mainly using QR codes. While QR codes offer a convenient way to link devices, they also present potential attack vectors [30]. In this paper, we focus on two attack scenarios involving QR codes in WhatsApp, WeChat, and Telegram:

(1) `QRSniffer-Browser`: In this scenario, malicious code executes without the user's knowledge, opening a browser window with "web.whatsapp.com." The code then captures a snapshot of the QR code and sends it to the attacker. Once the attacker scans the QR code, they can mirror the attacker's WhatsApp, WeChat, or Telegram account and inject messages.
(2) `QRSniffer-Prox`: An attacker near the user intercepts the QR code scanning process. When the user attempts to register a new device using the WhatsApp desktop app, WeChat, or Telegram, the attacker first uses a high-quality camera to scan the QR code. This results in the attacker's secondary device's application window displaying instead of the user's contacts.

These attack scenarios highlight severe security vulnerabilities that could affect millions of users across these popular E2EE messaging platforms. To address these vulnerabilities, we propose a novel short authentication string (SAS)- based authentication method, `SASLinker`, to link the secondary device and sync the account information. Unlike traditional QR code-based protocols, our SAS-based approach introduces a two way of verification during the linking device process. In addition, in a traditional SAS, both parties manually compare the authentication strings by saying the strings aloud; however, in our proposed mechanism, we mention getting the verification on the primary device and then confirming the verification on the secondary device.

In this approach, we proposed two mechanisms: a short authentication string pops up on the secondary device during the linking device mechanism. The primary device will verify that string [37], and the user gets the confirmation to accept or reject on the secondary device. Once the user accepts it, the linking device mechanism will be successful. Otherwise, it will reject the linking device mechanism. In the second scenario, we embed SAS in a QR code, and the secondary device displays that primary device. Once scanned and confirmed the verification, the secondary device gets the accept or reject, and once accepted, the device linking process will be successful; otherwise, it will be rejected. Notably, the SAS does not contain secret key material; it only authenticates keys, making it infeasible for an attacker to compromise the process simply by eavesdropping on the SAS. `SASLinker` enhances the security of E2EE messaging applications by introducing a significant layer of verification, thereby mitigating the risks associated with QR code-based attacks. This study examines the implementation of SAS-based authentication across WhatsApp, WeChat, and Telegram, evaluating its effectiveness in protecting user data and maintaining secure communication channels.

By addressing these security challenges, we aim to contribute to the ongoing efforts of the research community to enhance the security and privacy of E2EE messaging applications. This study seeks to answer critical questions: How can we improve the security of QR code-based linking device protocol? What are the most effective methods for preventing unauthorized access to user accounts? How can E2EE messaging applications balance convenience and security in their authentication processes? Through this research, we hope to provide valuable insights and practical solutions to enhance the security of instant messaging applications for millions of users worldwide.

**Our Contributions:** Our study elucidates the following contributions:

(1) We comprehensively analyze the existing linking device processes used by popular instant messaging applications such as WhatsApp, WeChat, and Telegram, identifying strengths, weaknesses, and potential security vulnerabilities related to QR code-based authentication.
(2) We demonstrate two attack scenarios that exploit these vulnerabilities: the `QRSniffer-Browser`, where malicious code captures and transmits a QR code to an attacker, and the `QRSniffer-Prox`, where an attacker intercepts the QR code scanning process to gain unauthorized access.
(3) We propose and implement the prototype of `SASLinker` to address these security issues. The SAS method displays a short authentication string on the secondary device during registration, which the user must enter on the primary device, adding an extra layer of security to prevent unauthorized access.

Through these contributions, our study aims to enhance the security of instant messaging applications by mitigating the risks associated with the current linking device mechanism.

## 2 Background

Understanding the intricacies of current secondary device linking protocols in widely adopted end-to-end encrypted (E2EE) messaging apps is essential to appreciate the significance of our proposed SAS-based device linking solution. This section provides an overview of these protocols and a review of Short Authentication String (SAS) authentication mechanisms.

### 2.1 Current Linking Device and Syncing Protocols for E2EE messaging apps

*2.1.1 WhatsApp:* WhatsApp employs a robust end-to-end encryption system based on the Signal Protocol, ensuring that third parties, including WhatsApp, cannot access plaintext messages or calls [41] [28]. The encryption keys are ephemeral and unique to each device, providing additional security even if a device's keys are compromised. During the linking device mechanism, a WhatsApp client transmits its public Identity Key, public Signed Pre Key, and a batch of public One-Time pre-keys to the server, which stores these keys associated with the user's identifier [14]. The primary device must create an Account Signature by signing the new device's public

Identity Key to link a secondary device. In contrast, the secondary device creates a Device Signature by signing the primary device's public Identity Key. This process involves scanning a QR code displayed on the secondary device using the primary device. The secondary device generates and displays a QR code containing its public Identity Key and a Linking Secret Key. The primary device scans the QR code, saves the secondary device's Identity Key, and generates necessary signatures and metadata. The server facilitates the secure transfer of linking data, ensuring both devices can establish end-to-end encrypted sessions. Upon linking a secondary device, the primary device end-to-end encrypts a copy of recent chat messages and transmits these to the secondary device [13]. This ensures that all messages remain encrypted during the syncing process, maintaining the confidentiality and integrity of the communication.

*2.1.2 WeChat and Telegram:* While the technical details of WeChat and Telegram's device linking protocols differ from WhatsApp, they share standard features, such as QR codes for linking secondary devices and end-to-end encryption to protect user communications. However, in the case of Telegram Login_Token, which is encoded using base64url, it is embedded in a $tg://login?token=base64encodedtoken$ URL and shown to the user as a QR code. In the case of WeChat, they share access_token, a base64url string encoded into the QR Code. Both platforms face similar vulnerabilities in their linking device processes, particularly regarding the exposure of QR codes to potential attackers [34] [39].

## 2.2 Review of SAS Authentication Protocols

Short Authentication String (SAS) protocols offer a promising solution to enhance the security of device-linking processes. SAS is a method to authenticate communication channels by comparing short, human-readable strings. This approach is commonly utilized in secure voice communication protocols, such as the ZRTP protocol in VoIP applications.

- *Principle of SAS*: SAS-based protocols generate a short string derived from the cryptographic material exchanged during the initial key agreement phase. This string is then displayed to both parties, who can verbally compare and verify it. If the strings match, the parties can be confident that their communication is secure and free from man-in-the-middle attacks.
- *Current Implementation*: Traditional entity authentication mechanisms, like those used in Signal (e.g., safety numbers), rely on long-term keys, which may not be secure if those keys are compromised [16]. The Signal protocol, widely used in applications like WhatsApp and Facebook Messenger, uses a combination of X3DH (Extended Triple Diffie-Hellman) for key agreement and the double ratchet algorithm for maintaining forward secrecy and post-compromise security. Signal's current entity authentication mechanism (safety numbers) relies on long-term keys, making them vulnerable if compromised. SAS-based authentication offers a robust alternative by allowing for shorter, more frequent out-of-band verifications.
- *Security Benefits*: The primary advantage of SAS-based authentication is that it does not transmit secret cryptographic material [27]. Instead, it uses the SAS for key verification, making it resistant to eavesdropping attacks. By adding this layer of verification, the protocol significantly enhances the security of the device-linking process without compromising user convenience [16].

In summary, while current linking device protocols in E2EE messaging apps like WhatsApp, WeChat, and Telegram provide robust security features, they are vulnerable to specific attack vectors involving QR code interception. Our proposed SAS-based authentication protocol addresses these vulnerabilities by introducing an added verification step that enhances security and maintains the integrity of user communications.

# 3 QRSniffer Attack Scenarios

This section explores the technical details of `QRSniffer-Browser` and `QRSniffer-Prox`, which exploit vulnerabilities in popular instant messaging application's QR code-based linking device processes. We conducted the experiment in a real WhatsApp account. These scenarios demonstrate the potential security risks users may face and highlight the need for improved authentication methods.


**Figure 1: `QRSniffer-Browser` Attack**

## 3.1 `QRSniffer-Browser`

**Feasibility and Testing:** The `QRSniffer-Browser` attack demonstrates the feasibility of a Remote Code Execution (RCE) exploit that compromises a user's instant messaging account, specifically WhatsApp, through user-space attacks. This attack bypasses the need for root access or compromises the kernel, leveraging user-space vulnerabilities instead. The testing process involved developing and executing malicious code designed to run silently in the background, automating the browser to capture and transmit QR codes.

**Attack Scenario:** In this scenario, the attacker begins by executing malicious code on the target device, often delivered through phishing emails, malicious attachments, or exploiting known software vulnerabilities. The code runs silently, opens the Chrome browser, and navigates to the WhatsApp Web login page. It captures the QR code displayed on the page using automation tools like Selenium or Puppeteer, sending the screenshot to the attacker's

server. The attacker then scans the QR code on their device, mirroring the attacker's WhatsApp account into the victim's secondary device and injecting a message to a random contact. Figure 1 depicts the detailed attack scenario.

**Practical Settings and Preferred Tools:** The practical settings for the `QRSniffer-Browser` attack involve two primary devices: the victim's device and the attacker's device. We select a device with a stable internet connection as the victim's device. The malicious code is executed on the victim's device, often via phishing emails, malicious attachments, or clicking a malicious web link. For our experiment, we sent a link from a device (referred to as the attacker's device) to the victim's device. When the user clicks the link, the victim's device opens the WhatsApp Web link. The attacker's device receives and scans the captured QR code, mirroring the attacker's WhatsApp account. Preferred tools for this attack include Selenium WebDriver, which automates browser actions such as opening the WhatsApp Web page and capturing the QR code. Secure HTTP requests are used to avoid detection by network monitoring tools and ensure the QR code is transmitted securely.

**Implementation Detail:** The attack begins with setting up and configuring Selenium WebDriver for Chrome to automate the process on the victim's device. The browser navigates to the WhatsApp Web login page, waiting for the QR code to appear and ensuring the page is fully loaded. Upon displaying the QR code, automation scripts capture a screenshot, which is transmitted to the attacker's server using secure HTTP requests. The attacker then scans the QR code, mirroring their WhatsApp account onto the victim's device.

To propagate the attack, the automation identifies and selects a random contact from the contact list, accesses the chat window, and sends a pre-defined malicious message. The automation ensures seamless execution, highlighting the ease with which such vulnerabilities can be exploited. This process demonstrates the effectiveness of remote code execution attacks through automation. The complete flow of the attack is illustrated in Figure 2, showcasing how critical vulnerabilities in device linking protocols can be leveraged maliciously.

**Attack Demonstration:** We conducted a demonstration of the designed attack at: https://sites.google.com/view/qr-sniffer-attack-demo/home

**Consequences of `QRSniffer-Browser` Attack :** As demonstrated above, the attacker's ability to manipulate the victim's text messages is a serious concern. By mirroring their WhatsApp account onto the victim's device, the attacker gains more accessibility to the victim's messages, potentially showing obscene messages or altering the content.

It is crucial to understand the potential harm that an attacker can cause. In Figure 6, we show the type of obscene messages the attacker can send through the victim's device. As the victim does not know whether the secondary device contains the actual account they linked, the victim might think they are sending text messages to the contacts. In addition, text messages are written and sent automatically, making it easier to believe that someone is taking over the victim's account and asking for ransom money, as depicted in Figure 4. The attacker's motivation likely revolves around exploiting the victim's device as a platform for spreading malicious content or causing confusion and distress. By mirroring their WhatsApp account onto the victim's device, the attacker can

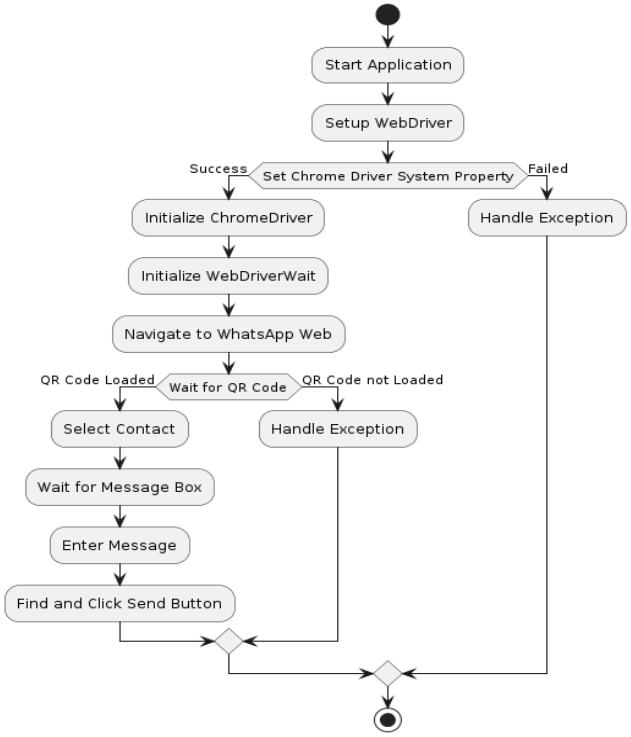


**Figure 2: Flowchart of the malicious code**

send sexually explicit or spam messages from their account but from the victim's device. This could lead to several damaging outcomes for the victim. The sexually explicit messages or links provided in the message depicted in Figure 3 might be used by the attacker to shock or embarrass the victim by making it appear as though their device is responsible for sending inappropriate content. In the case of spam messages depicted in Figure 5, the attacker could be attempting to use the victim's device as a relay to distribute malicious links or spam links providing false offers to their contacts, thereby avoiding detection or filtering systems that might block such content if sent directly from the attacker's device.

Another scenario describes a potential case where the attacker's account gets mirrored into the victim's secondary device. This case can easily trick the victim as the victim links their device for the first time by adding the contact. Once the user adds a contact to the secondary device, they add that contact to the attacker's account, a severe violation of the End-to-End encryption scheme. In Figure 9, we can observe an empty contact list of the attacker's account, and the victim adds a contact to that chat window without knowing they are adding that to the attacker's account. Although the user account has not been compromised, the fact that their device is being used to mirror and propagate these harmful messages can lead to significant reputational damage, as the victim may be complicit in these actions. Additionally, the victim might experience a loss of trust in their digital security, leading to increased anxiety about their data and communications. This scenario highlights the sophisticated nature

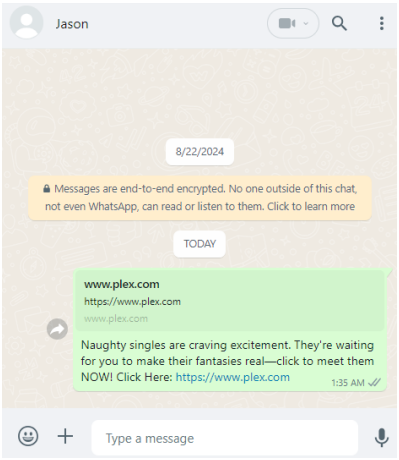


Figure 3: Sexually Explicit Message

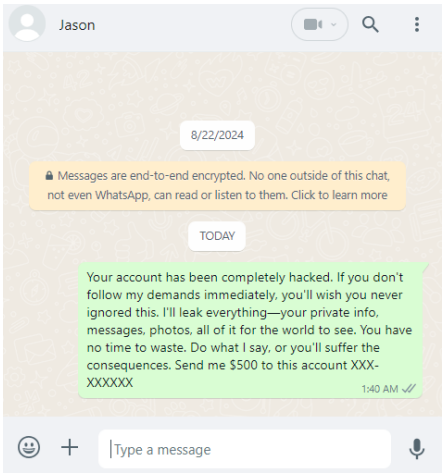


Figure 4: Harassment or Threatening Message

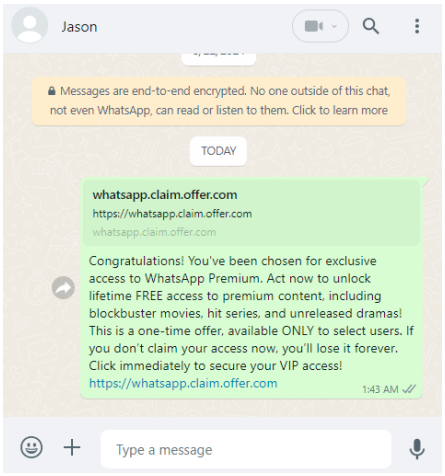


Figure 5: Inappropriate or Spam Messages

Figure 6: Obscene Messages Sent from the Victim's Secondary Device for WhatsApp

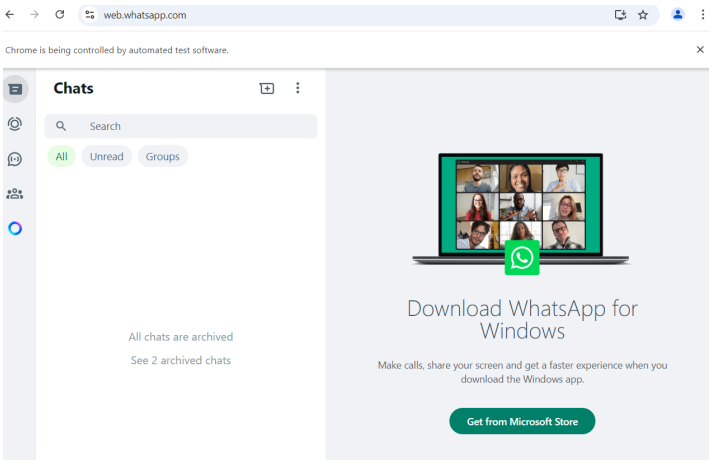


Figure 7: Device Linked with No Contact

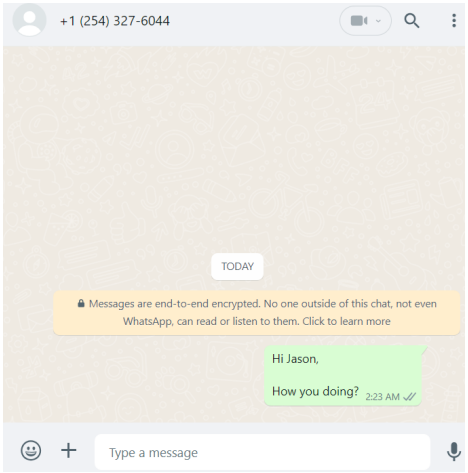


Figure 8: Victim Adds Contact to the Attacker's Account

Figure 9: Inducing a Victim to Unknowingly Add a Malicious Contact

of modern cyber threats and the importance of vigilance and robust security practices to protect against such attacks.

## 3.2 QRSniffer-Prox

QRSniffer-Prox exploits the physical proximity of the attacker to the victim during the linking device process of the desktop application. This attack allows an adversary to capture and use the QR code displayed on the victim's screen; thereby, the victim's secondary device gains access to the attacker's malicious account. For our experiment, we used WhatsApp and a device where the desktop WhatsApp application was already installed, referred to as the victim's device.

**Feasibility and Testing:** To assess the feasibility of this attack, we conducted tests using various camera setups. Our tests found that the optimal distance to capture the QR code with a high-quality smartphone camera is approximately 8 feet 5 inches. Using a 15x zoom ensures the QR code is clear and readable. This configuration allows for precise and effective QR code capture without alerting the target. Using higher-quality cameras with advanced lenses can significantly enhance the effectiveness of this attack. Better cameras with superior optical zoom capabilities can capture the QR code from even greater distances while maintaining clarity and

readability. For instance, a camera with a 30x or higher optical zoom could capture the QR code from distances exceeding 15 feet. This extended range increases the attacker's flexibility in positioning the camera, making it more challenging for the target to detect the surveillance.

**Attack Scenario:** The attacker positions themselves close to the victim, such as in a public place or shared office environment, where they can observe the victim attempting to register a new device using the WhatsApp desktop app, WeChat, or Telegram. This proximity allows the attacker to view the victim's device screen during the linking device process. The attacker discreetly uses the camera setup described above to capture the QR code.

As the victim attempts to link their device, the attacker screens the camera to scan the QR code before the victim completes registration. This precise timing and use of advanced camera equipment ensure the clarity and accuracy of the captured QR code. The attacker's account may initially show an empty contact list on the victim's secondary device. However, once the QR code is scanned, the attacker's messaging service account mirrors into the victim's device, granting access to read, send, and access the contact list. To avoid detection, the attacker can manipulate the IM application's interface on their device to display an empty contacts list, making it appear that the user is accessing their account and adding a contact.

**Practical Settings:** The practicality of this attack is particularly high in shared office settings or public places where people might link their devices. For example, colleagues might work nearby in an open office environment, making it easier for an attacker to position a camera discreetly. Similarly, individuals often link their devices while unaware of their surroundings in public places like cafes or libraries, providing an opportunity for attackers [1].

**Implementation Details:** Using a smartphone camera positioned approximately 8 feet 5 inches away with 15x zoom, we demonstrated an attack exploiting the WhatsApp desktop linking process. Using an iPhone 13 Pro Max with a 13mm f/1.8-aperture lens, PDAF, and 2cm macro, the QR code was effectively captured in a closed setting. Once scanned, the attacker gained full access to the victim's account. This highlights the need for stronger security measures to ensure only authorized devices can link. Advanced cameras with higher optical zoom and superior lenses could enhance the attack, capturing QR codes from greater distances with improved clarity and stealth.

The attack scenario is depicted in Figure 10. When using the desktop WhatsApp application for the first time on an operating system, the user needs to scan the QR code displayed on their device. During this process, an attacker can position a camera up to 8.5 feet away, utilizing a 15x zoom to scan the QR code before the user completes their scan.

The attacker exploits the device-linking process, which typically takes 12 to 15 seconds, as the user navigates to the "Linked Devices" option on their primary device and activates the camera to scan the QR code. During this window, the attacker captures the QR code, causing an empty contact list and chatbox to appear on the user's device. The attacker deceives the user into believing they are interacting with their account. When the user adds a new contact, the attacker gains access to the contact information, compromising security.

**Consequences of Device Linking by an Attacker:** As a consequence of this attack, the victim, deceived by the empty contact list displayed during the initial linking process, unknowingly adds new contact information to the attacker's account. This action compromises the end-to-end encryption scheme, exposing sensitive user data and undermining the platform's security integrity.

# 4 SASLinker: Our Proposed Defense

## 4.1 Overview

The Short Authentication String (SAS)–based authentication method [16] enhances the device's linking process's security for E2EE messaging apps like WhatsApp, WeChat, and Telegram. This method introduces a robust verification step that ensures only legitimate users can complete the registration of a secondary device, thereby protecting against unauthorized access.

We propose SASLinker a solution that securely links secondary devices to primary devices without directly exchanging any shared keys or identities. This mechanism relies on a combination of secure commitment schemes and cryptographic techniques to verify the devices' authenticity.

Our proposed defense mechanism uses the Short Authentication Strings (SAS) approach to enable secure communication over an insecure channel via a narrowband channel for SAS authentication. The process starts when the sender (Alice) creates a commitment value by combining a message and a random string. Alice sends this commitment to the recipient (Bob) over the insecure channel. Bob then generates and sends a random string to Alice, who reveals her random string to open the commitment. Bob verifies the commitment's integrity, and both parties calculate the SAS as the XOR of their random strings. The SAS is sent through the secure narrowband channel, ensuring integrity and authenticity even in an insecure communication environment.

The SAS mechanism resists man-in-the-middle (MitM) attacks, as any interference alters the SAS, making such attacks easily detectable. Cryptographic commitment schemes ensure that once a value is committed, it cannot be modified without detection. This makes SAS reliable when public key infrastructures are unavailable or compromised, such as peer-to-peer networks or secure device pairing for Bluetooth, SSH, and PGP.

When a user initiates linking a secondary device (SD) to their primary device (PD), the process starts with the SD generating a Diffie-Hellman (DH) key pair and a random value [11]. The SD computes a commitment based on this information and sends the commitment and its DH public key to the server. The server then forwards these to the PD. Upon receiving them, the PD generates its own DH key pair and a random value, computes its commitment, and sends the commitment and DH public key back to the server, which forwards them to the SD. Both devices hold each other's obligations and DH public keys at this stage without revealing the underlying random values.

The PD and SD exchange their random values through the server. This exchange is crucial for the verification step, ensuring both devices have all the necessary information. Each device then verifies the commitment received from the other device using the exchanged random values. This step ensures that the commitments are authentic and have not been tampered with.

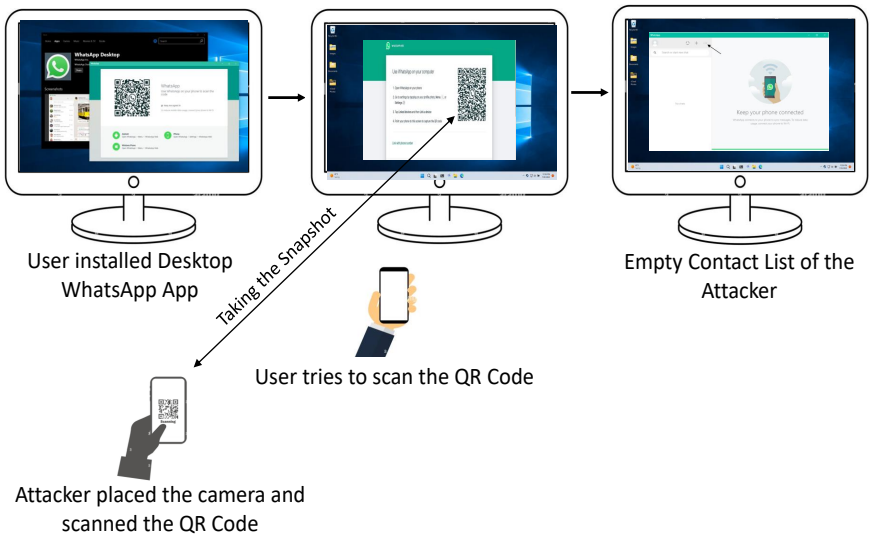


**Figure 10: `QRSniffer-Prox` Attack**

After successful verification, both devices independently compute a shared key using the DH public and private keys. In this process, no keys are transferred between each other. They then compute a Short Authentication String (SAS) using the shared key and the exchanged random values. The SAS is a unique string derived from these values. The SAS is computed by taking the first 16 bytes of the shared secret, converting them to an integer, and then performing bitwise XOR operations with two random values (also converted to integers). The result is converted to a hexadecimal string. This process ensures a unique and verifiable SAS for secure device linking. There are potentially two ways the SAS can be handled:

(1) The alphanumeric SAS can be displayed on the secondary device, and the PD user needs to enter the displayed alphanumeric SAS. If the SAS matches the generated SAS on the PD, it sends the confirmation to the SD, and the user can accept or reject the confirmation displayed on the SD.
(2) The SAS is embedded into a QR code displayed on the SD, and the PD scans the QR code displayed on the SD. Once the SAS is verified on the PD, the PD sends the confirmation to accept or reject it to the secondary device.

If the user accepts it, the linking device mechanism is successful; otherwise, it will reject the linking procedure. This step completes the linking process, ensuring the SD is securely linked to the PD. By leveraging the SAS-based approach, this method ensures secure and user-friendly linking of secondary devices, mitigating the risk of unauthorized access by requiring physical presence and direct interaction between the devices.

## 4.2 Detailed Explanation of How `SASLinker` Addresses Specific Vulnerabilities

The proposed SAS-based authentication method, `SASLinker` enhances the security of the device linking process for instant messaging applications, effectively mitigating the attacks we discussed. Here's how the algorithm addresses these specific attacks:

**The `QRSniffer-Browser` Attack:** In this attack scenario, malicious code executes without the user's knowledge, opening a browser window with "web.whatsapp.com." The code captures a snapshot of the QR code and sends it to the attacker, allowing them to mirror the victim's account.

**Mitigation by `SASLinker`:**

(1) Commitment and Verification: The initial steps of generating and exchanging commitments between the primary device (PD) and secondary device (SD) ensure that both devices have cryptographic information that needs to be verified. Even if the attacker captures the QR code or the alphanumeric SAS, they cannot generate the valid commitments or the corresponding random values required for the verification process.
(2) SAS Computation: Both devices compute the SAS independently using their verified cryptographic information. The SD displays a QR code containing the SAS where the attacker cannot replicate the process since the SAS is derived from secret random values never transmitted directly.
(3) Terminating the Malicious Code Execution: The two-way confirmation process significantly hinders an attacker from interfering with the acceptance or rejection of the SAS. Since the secondary device must receive and process the confirmation to accept or reject the SAS, an attacker cannot manipulate this step using malicious code or through their primary device. This ensures that only the legitimate user can complete the linking process.

**The QRSniffer-Prox Attack:** In this attack scenario, an attacker intercepts the QR code scanning process by using a high-quality camera to scan the QR code before the legitimate user, resulting in the attacker's IM window displaying instead of the user's contacts.

**Mitigation by SASLinker:**

(1) Commitment and Verification: Similar to the previous attack, the commitment generation and verification steps ensure that only devices with matching commitments and corresponding random values can successfully compute the SAS. The attacker cannot obtain these values merely by capturing the QR code or the alphanumeric SAS.
(2) SAS Computation: The SAS is computed based on verified cryptographic information, ensuring its uniqueness and integrity. The SD displays a QR code containing the SAS only after these values have been securely verified. An attacker cannot reproduce or use the captured QR code or the value displayed on the SD. Since no key is shared during this procedure, the attacker cannot make any progress with the attack.
(3) Physical Presence Requirement: The QR code contains the SAS, which the PD scans. Suppose an attacker captures the QR code or the SAS displays on the SD with a high-quality camera and attempts to use it. In that case, they will not have the necessary cryptographic values to generate a valid SAS on their device. Additionally, the legitimate user will notice that the secondary device will get the confirmation about accepting or rejecting.
(4) Secure Environment: The linking process requires physical presence and direct interaction between the SD and PD. This requirement makes it significantly harder for an attacker to intercept the QR code without being detected.

By leveraging the SAS-based approach and commitment verification, SASLinker provides a robust defense against these QR code-based attacks. The mechanism ensures that even if an attacker captures the SAS or the QR code, they cannot complete the device-linking process without possessing the correct cryptographic information and physical access to both devices. This method significantly reduces the risk of unauthorized access and ensures the security of the device linking mechanism.

## 4.3 Authentication Methods Comparison

**Password-Based Authentication:** Password-based authentication relies on users creating and remembering passwords stored in the hashed form on servers and compared during login. This method is vulnerable to brute force attacks, dictionary attacks, and phishing, especially when users create weak or reused passwords [26]. Users often need help creating and remembering strong passwords, leading to insecure practices like writing them down or reusing them across services. In contrast, the SAS-based method (SASLinker) uses cryptographic techniques like commitments and Diffie-Hellman key exchange, avoiding the need to store or transmit passwords. This approach enhances security by ensuring the process cannot be tampered with and improves usability by involving simple actions like scanning a QR code or entering a Short Authentication String (SAS).

**Biometric Authentication:** Biometric authentication relies on unique physical traits, such as fingerprints, facial recognition, or iris scans, stored securely on devices or sometimes on servers. During authentication, captured biometric data is processed and compared to the stored data. While it provides high security, biometric data can still be spoofed or stolen if not adequately protected. Key handling in biometric systems involves securely storing biometric data on devices or servers, requiring robust protection against unauthorized access [5]. Biometric templates must be encrypted and securely managed to prevent leaks and misuse. SASLinker avoids storing sensitive biometric data using DH key pairs and commitments, ensuring robust security against tampering and offering a user-friendly linking process through QR codes or SAS. This method avoids the privacy concerns associated with biometric data storage. Key handling in SASLinker focuses on using DH key pairs and commitments, preventing the need to store sensitive biometric data and emphasizing cryptographic security without the complexities of managing biometric templates.

**Public Key Infrastructure (PKI):** Public Key Infrastructure (PKI) employs a pair of cryptographic keys for secure communication and user authentication, involving the secure management of private keys on devices and public keys on a certificate authority (CA) server. During authentication, a challenge-response mechanism is used, where the user signs a challenge with their private key, and the server verifies the signature using the public key. While PKI provides a high level of security through asymmetric encryption, managing and distributing certificates can be complex and cumbersome, requiring technical knowledge [7]. SASLinker simplifies key management by using DH key pairs and commitments, offering significant security without the complexity of certificate management. It provides a more user-friendly approach using QR codes or SAS for device linking, making it accessible to non-technical users while maintaining strong security.

# 5 SASLinker: Design and Implementation

This section provides a detailed implementation of the Short Authentication String (SAS)–based authentication method for enhancing the security of linking device processes in End applications such as WhatsApp, WeChat, and Telegram. The implementation encompasses the SAS generation, display, and verification during the linking device process.

## 5.1 System Architecture

The system architecture for SAS-based authentication involves several key components, including the IM application server and primary and secondary devices. The interaction between these components is crucial for ensuring a secure linking device process. Figure 11 illustrates the overall architecture and interaction flow.

## 5.2 Implementation Detail

(1) **Generate and Commit Keys (SD):** The Secondary Device (SD) generates a Diffie-Hellman (DH) key pair $(d_{SD}, g^{d_{SD}})$ and a random value $R_{SD}$. It computes the commitment $C_{SD} = \text{commit}(R_{SD}, g^{d_{SD}})$ and sends $C_{SD}$ and $g^{d_{SD}}$ to the server.
(2) **Generate and Commit Keys (PD):** The Primary Device (PD) generates its own DH key pair $(d_{PD}, g^{d_{PD}})$ and random

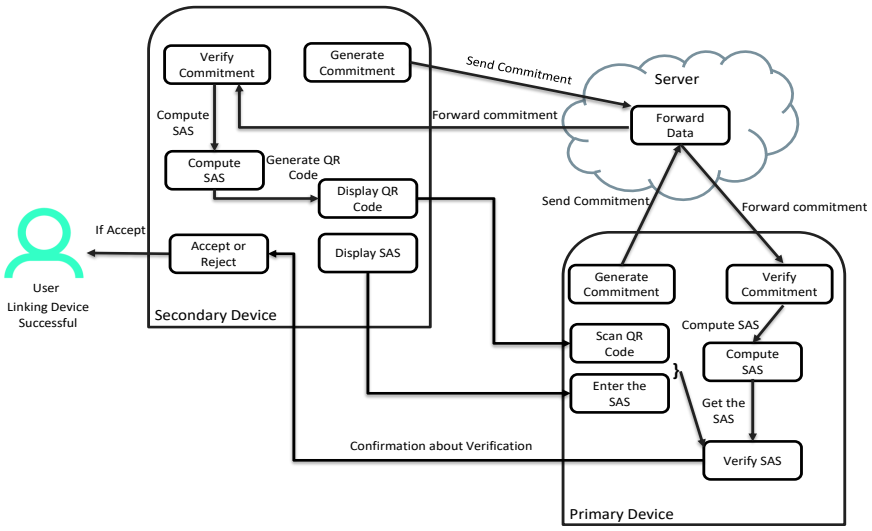


**Figure 11: System Architecture for SASLinker**

value $R_{PD}$. It computes the commitment $C_{PD}$ = commit $(R_{PD}, g^{d_{PD}})$.

(3) **Exchange Commitments and Public Keys:** The server forwards $C_{SD}$ and $g^{d_{SD}}$ to the PD, and $C_{PD}$ and $g^{d_{PD}}$ to the SD. The server acts as an intermediary that facilitates the forwarding process.
(4) **Exchange Random Values:** The PD and SD exchange $R_{PD}$ and $R_{SD}$ via the server.
(5) **Verify Commitments:** The PD verifies $C_{SD}$ using $R_{SD}$ and $g^{d_{SD}}$, and the SD verifies $C_{PD}$ using $R_{PD}$ and $g^{d_{PD}}$.
(6) **Compute Shared Keys:** The PD computes the shared key $K_{PD} = (g^{d_{SD}})^{d_{PD}}$, and the SD computes $K_{SD} = (g^{d_{PD}})^{d_{SD}}$.
(7) **Compute and Verify SAS:** Both devices compute the Short Authentication String (SAS) using $K_{PD}$, $R_{PD}$, and $R_{SD}$. The SD displays the SAS as either a QR code or a string. The PD scans or manually enters the SAS, verifies it, and sends a confirmation to the SD. If the SAS matches, the SD user accepts the confirmation to complete the linking mechanism; otherwise, it fails.

The detailed procedures and algorithms for implementing the SASLinker mechanism, including the SAS-Based Device Linking Protocol and SAS computation, are provided in the Appendix. These include the step-by-step generation, exchange, and verification of cryptographic keys and calculating the Short Authentication String (SAS), ensuring secure device linking in IM applications. In addition, the Appendix illustrates the overall implementation details, highlighting how the mechanism mitigates potential attacks, such as intercepting QR codes, while maintaining the integrity of the linking process.

## 6 Discussion

The research conducted in this paper aimed to address significant vulnerabilities in the device-linking protocols of widely used end-to-end encrypted (E2EE) messaging applications like WhatsApp, WeChat, and Telegram. The study proposed and evaluated a SAS-based authentication mechanism, SASLinker, to mitigate the risks posed by QR code-based attacks, specifically QRSniffer-Browser and QRSniffer-Prox.

The process involved a detailed analysis of the current linking protocols and identifying their weaknesses, particularly the reliance on QR codes for transmitting critical cryptographic material. The introduction of SASLinker sought to overcome these vulnerabilities by incorporating a short authentication string (SAS) into the device linking process, adding a layer of security.

We evaluate the effectiveness of SASLinker in addressing the questions posed in the introduction:

**Improving the Security of QR Code-based Linking Device Protocol:** SASLinker introduces a robust mechanism that significantly enhances the security of the linking device process. Requiring users to validate a short authentication string ensures that even if an attacker captures the QR code, they cannot proceed without the SAS, as no secondary device identification is added to the QR code. This method effectively mitigates the risk of unauthorized access through QR code interception.

**Preventing Unauthorized Access to User Accounts:** The SAS-based approach requires physical presence and direct interaction between the primary and secondary devices, making it difficult for attackers to execute QRSniffer attacks. The commitment and verification steps ensure that only legitimate devices of the user can complete the linking process, thereby preventing unauthorized access.

**Balancing Convenience and Security:** While adding an extra step to the authentication process, SASLinker maintains user convenience using a straightforward copying alphanumeric SAS code or QR code scanning method and SAS verification. This approach balances between enhancing security and preserving the ease of use users expect from E2EE messaging applications.

Overall, the proposed SASLinker mechanism addresses the identified vulnerabilities effectively and provides a practical solution

for improving the security of E2EE messaging applications. The study's findings contribute to the ongoing efforts to enhance the protection and privacy of user data in the digital communication landscape.

## 7 Related Work

Recent research has significantly advanced our understanding of remote code execution (RCE) vulnerabilities and their impact on device security. Wichelmann et al. [40] investigated the vulnerabilities in Signal's multi-device protocol, Sesame, which allows attackers to register a malicious device and compromise future communications. It revealed that the current implementation does not guarantee post-compromise security, as attackers can stealthily add devices, gaining unrestricted access to all future communications and impersonating the victim. Campion et al. propose a new protocol for secure multi-device communication based on the Signal protocol, addressing the limitations of existing solutions like Sesame. The authors introduce a Ratcheted Dynamic Multicast (RDM) protocol to securely synchronize keys across multiple devices without revealing the device count or identity, enhancing privacy and efficiency [12]. Despite using advanced encryption, Bahramal et al. [9] and Bogos et al.[10] demonstrate that popular Instant Messaging (IM) applications like Telegram, Signal, and WhatsApp are vulnerable to traffic analysis attacks.

Isobe et al. [20] identify vulnerabilities in LINE's Letter Sealing E2EE scheme, allowing forgery and impersonation attacks by an end-to-end adversary. They propose countermeasures to enhance security, which LINE Corporation plans to implement. Jain et al. [21] identify security issues in a Signal-based messaging app, including session hijacking and media jacking. The authors use STRIDE threat modeling and report their findings to improve the app's security. Hindocha and Chien [18] discuss various malicious threats to instant messaging clients, including backdoor Trojan horses that provide unauthorized remote access to hackers. Their study on vulnerabilities and blended threats in IM applications reveals how they can propagate rapidly through instant messaging networks, often exploiting weak spots in the client software to gain control over devices. Their findings emphasized the need for robust defenses against cross-context execution (Xrce) attacks, which can exploit these vulnerabilities to run arbitrary executables on victim devices. These studies collectively provide valuable insights into the security challenges and solutions for defending against remote code execution attacks.

Understanding the nuances of QR code security has become increasingly critical in the face of sophisticated threats like the Proximity QR Code Scanning Attack. Wahsheh and Al-Zahrani [38] delved into the intricacies of detecting malicious URLs embedded in QR codes using advanced computational intelligence models, including fuzzy logic and multilayer perceptron artificial neural networks (MLP-ANN). Their research underscores the inherent vulnerabilities in QR code generation and the importance of real-time detection mechanisms to thwart unauthorized access attempts via QR codes. Similarly, Sahay et al. [30] investigated the security challenges in SaaS environments, such as SQL injection and collision attacks, stressing the necessity of secure QR code generation and the application of machine learning techniques to identify and mitigate malicious activities. Their work highlights the critical role of robust security practices in preventing exploitation through QR codes.

Furthermore, Al-Zahrani et al. [4] presented a secure artificial intelligence system to detect malicious QR codes, employing various AI models to identify harmful links. This research emphasizes the need for sophisticated detection methods to safeguard against advanced attacks like the Proximity QR Code Scanning Attack, where attackers intercept and misuse QR codes to access sensitive user information. These studies collectively provide a comprehensive foundation for enhancing QR code security and developing defenses against proximity-based interception attacks.

Alatawi and Saxena [6] analyzed end-to-end encryption (E2EE) and authentication ceremonies in secure messaging systems, identifying significant vulnerabilities in existing E2EE apps, especially against man-in-the-middle (MitM) attacks. Their study underscores the importance of robust authentication protocols to prevent unauthorized access and ensure secure communications. Similarly, Rodrigues et al. [29] explored hardware-based cryptography and authentication for securing instant messages, emphasizing the critical need for robust encryption and secure device authentication methods to protect against remote code execution and message injection attacks. Ahn et al. [3] proposed a unified framework for end-user authentication protocols in Feature-as-a-Service models, highlighting the necessity of standardized authentication methods to secure user interactions against sophisticated attacks involving QR code vulnerabilities. Building on these insights, we propose a Short Authentication String (SAS)-based linking device process to address these vulnerabilities. This method displays a short authentication string on the secondary device during registration, which the user enters on the primary device. This dual-device verification ensures secure authentication and registration, effectively mitigating the risks associated with remote code execution and QR code scanning attacks.

## 8 Conclusion

The research conducted in this paper addresses critical vulnerabilities in the device-linking protocols of widely adopted end-to-end encrypted (E2EE) messaging applications such as WhatsApp, WeChat, and Telegram. The study highlights the inherent risks of current QR code-based authentication methods, demonstrating how these vulnerabilities can be exploited through `QRSniffer-Browser`and `QRSniffer-Proxa`ttacks.

We introduced `SASLinker` a Short Authentication String (SAS)-based authentication mechanism that mitigates these threats and enhances security by requiring users to validate an SAS during the linking mechanism of a secondary device. This prevents unauthorized access even if a QR code is intercepted, offering a practical solution that maintains user convenience while significantly improving security through direct interaction and verification steps. While adding steps to the authentication process, `SASLinker` maintains user convenience through simple QR code scanning or alphanumeric SAS verification, balancing the need for enhanced security with ease of use.

Our study provides a comprehensive analysis of existing protocols, demonstrates the vulnerabilities, and presents the `SASLinker`

solution, which significantly enhances the security of device-linking processes in E2EE messaging apps. By addressing these challenges, `SASLinker` ensures the privacy and security of millions of users worldwide, contributing valuable insights and solutions to the research community.

## References


[1] Mohamad Abdulkader. 2023. Why do people use public Wi-Fi?: An investigation of risk-taking behaviour and factors lead to decisions.

[2] Asmara Afzal, Mehdi Hussain, Shahzad Saleem, M Khuram Shahzad, Anthony TS Ho, and Ki-Hyun Jung. 2021. Encrypted network traffic analysis of secure instant messaging application: A case study of signal messenger app. *Applied Sciences* 11, 17 (2021), 7789.

[3] Jaehyung Ahn, Junhong Min, Hyung Tae Lee, and Jeongyeup Paek. 2023. Unified Framework for End-User Authentication Protocol in Feature-as-a-Service Models. In *2023 14th International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 539–542.

[4] Mohammed S Al-Zahrani, Heider AM Wahsheh, and Fawaz W Alsaade. 2021. Secure real-time artificial intelligence system against malicious QR code links. *Security and Communication Networks* 2021 (2021), 1–11.

[5] Abdulmonam Omar Alaswad, Ahlal H Montaser, and Fawzia Elhashmi Mohamad. 2014. Vulnerabilities of biometric authentication threats and countermeasures. *International Journal of Information & Computation Technology* 4, 10 (2014), 947–58.

[6] Mashari Alatawi and Nitesh Saxena. 2023. SoK: An Analysis of End-to-End Encryption and Authentication Ceremonies in Secure Messaging Systems. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 187–201.

[7] Ohoud Albogami, Manal Alruqi, Kholood Almalki, and Asia Aljahdali. 2021. Public key infrastructure traditional and modern implementation. *International Journal of Network Security* 23, 2 (2021), 343–350.

[8] Kirill Arbuzov. 2023. Advanced spyware for mobile devices. (2023).

[9] Alireza Bahramali, Ramin Soltani, Amir Houmansadr, Dennis Goeckel, and Don Towsley. 2020. Practical traffic analysis attacks on secure messaging applications. *arXiv preprint arXiv:2005.00508* (2020).

[10] Corina-Elena Bogos, Răzvan Mocanu, and Emil Simion. 2023. A security analysis comparison between Signal, WhatsApp and Telegram. *Cryptology ePrint Archive* (2023).

[11] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. 2007. Provably secure authenticated group Diffie-Hellman key exchange. *ACM Transactions on Information and System Security (TISSEC)* 10, 3 (2007), 10–es.

[12] Sébastien Campion, Julien Devigne, Céline Duguey, and Pierre-Alain Fouque. 2020. Multi-device for signal. In *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part II 18*. Springer, 167–187.

[13] Tom Carpay and Pavlos Lontorfos. 2019. WhatsApp End-to-End Encryption: Are Our Messages Private? *Retrieved* 2, 05 (2019), 2020.

[14] Facebook. 2024. Document from Facebook. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf Accessed: 2024-05-26.

[15] Oluwatoyin Ajoke Farayola, Oluwabukunmi Latifat Olorunfemi, and Philip Olaseni Shoetan. 2024. Data privacy and security in IT: a review of techniques and challenges. *Computer Science & IT Research Journal* 5, 3 (2024), 606–615.

[16] Sébastien Hauri. 2022. SAS-based authentication for secure messaging. (2022).

[17] Amir Herzberg and Hemi Leibowitz. 2016. Can Johnny finally encrypt? Evaluating E2E-encryption in popular IM applications. In *Proceedings of the 6th Workshop on Socio-Technical Aspects in Security and Trust*. 17–28.

[18] Neal Hindocha and Eric Chien. 2003. Malicious threats and vulnerabilities in instant messaging. In *Virus Bulletin Conference, vb2003*.

[19] Johannes Hönlinger. 2018. The role of instant messenger as computermediated communication tool for knowledge sharing and teamwork performance.

[20] Takanori Isobe and Kazuhiko Minematsu. 2018. Breaking message integrity of an end-to-end encryption scheme of LINE. In *Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II 23*. Springer, 249–268.

[21] Kartikeya Jain, Anagha Ananth, and Prasad Honnavalli. 2021. Vulnerability Analysis of a Signal-based Messenger. In *2021 IEEE Bombay Section Signature Conference (IBSSC)*. IEEE, 1–6.

[22] Kehinde Funmilayo Mefolere. 2016. WhatsApp and information sharing: Prospect and challenges. *International Journal of Social Science and Humanities Research* 4, 1 (2016), 615–625.

[23] Leah Moyle, Andrew Childs, Ross Coomber, and Monica J Barratt. 2019. # Drugsforsale: An exploration of the use of social media and encrypted messaging apps to supply and access drugs. *International Journal of Drug Policy* 63 (2019), 101–110.

[24] Sunetra Sen Narayan and Shalini Narayanan. 2024. *The WhatsApp India Story: Inside the Digital Maya Sphere*. Taylor & Francis.

[25] Rolf Oppliger. 2020. *End-to-end Encrypted Messaging*. Artech House.

[26] Jim Owens and Jeanna Matthews. 2008. A study of passwords and methods used in brute-force SSH attacks. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*. 8.

[27] Riccardo Pecori and Luca Veltri. 2016. 3AKEP: Triple-authenticated key exchange protocol for peer-to-peer VoIP applications. *Computer Communications* 85 (2016), 28–40.

[28] Nidhi Rastogi and James Hendler. 2017. WhatsApp security and role of metadata in preserving privacy. *arXiv Prepr. arXiv1701* 6817 (2017), 269–275.

[29] Gabriel Arquelau Pimenta Rodrigues, Robson De Oliveira Albuquerque, Gabriel De Oliveira Alves, Fábio Lúcio Lopes De Mendonça, William Ferreira Giozza, Rafael Timóteo De Sousa, and Ana Lucila Sandoval Orozco. 2020. Securing instant messages with hardware-based cryptography and authentication in browser extension. *IEEE Access* 8 (2020), 95137–95152.

[30] Manushree Sahay, Sandeep Vanjale, and Madhavi Mane. 2024. Software As Service Attack Detection and Prevention for Deceitful QR code. *International Journal of Intelligent Systems and Applications in Engineering* 12, 4s (2024), 454–462.

[31] Shivangi Singh. 2024. PEGASUS SPYWARE AND RIGHT TO PRIVACY IN INDIA. (2024).

[32] Thomas Strasser. 2020. App, app'n'away. How social messaging tools like WhatsApp support mobile language learning and teaching. *heiEDUCATION Journal. Transdisziplinäre Studien zur Lehrerbildung* 5 (2020).

[33] Belén Suárez-Lantarón, Yolanda Deocano-Ruíz, Nuria García-Perales, and Irina Sherezade Castillo-Reche. 2022. The educational use of WhatsApp. *Sustainability* 14, 17 (2022), 10510.

[34] Telegram. 2024. Telegram QR Login. https://core.telegram.org/api/qr-login Accessed: 2024-05-26.

[35] V Thangavel. 2024. Nomophobia in India: A psychological disorder that causes the brain to release dopamine in response to tweets, emoticons, and other acts, rewarding the behavior and sustaining the habit of using social media addiction. *Curr Trends Mass Comm* 3, 1 (2024), 01–16.

[36] Amin Tootoonchian, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. 2009. Lockr: better privacy for social networks. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. 169–180.

[37] Serge Vaudenay. 2005. Secure communications over insecure channels based on short authenticated strings. In *Annual International Cryptology Conference*. Springer, 309–326.

[38] Heider AM Wahsheh and Mohammed S Al-Zahrani. 2021. Secure real-time computational intelligence system against malicious QR code links. *International Journal of Computers Communications & Control* 16, 3 (2021).

[39] WeChat Developers. 2024. Login via Scan. https://developers.weixin.qq.com/doc/oplatform/en/Mobile_App/WeChat_Login/Login_via_Scan.html Accessed: 2024-05-26.

[40] Jan Wichelmann, Sebastian Berndt, Claudius Pott, and Thomas Eisenbarth. 2021. Help, My Signal has Bad Device! Breaking the Signal Messenger's Post-Compromise Security Through a Malicious Device. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 18th International Conference, DIMVA 2021, Virtual Event, July 14–16, 2021, Proceedings 18*. Springer, 88–105.

[41] Fietyata Yudha, Ahmad Luthfi, and Yudi Prayudi. 2017. A proposed model for investigating on web WhatsApp application. *Advanced Science Letters* 23, 5 (2017), 4050–4054.


## A Algorithms for SAS-Based Device Linking and SAS Computation

We developed two algorithms to describe the procedures for the above mechanism. The SAS-Based Device Linking Algorithm 1 ensures secure pairing between a Secondary Device (SD) and a Primary Device (PD) using Diffie-Hellman key exchange and commitment schemes. The devices exchange commitments and random values, compute a shared key, and verify a Short Authentication String (SAS) to complete the secure linking process. Our approach comprehensively provides a better verification step than comparison-based manual verification of SAS, as we provide two ways of confirmation and verification.

Algorithm 2 computes SAS by processing a shared secret and two random values. It ensures a unique, verifiable string for secure device pairing. The algorithm employs bitwise XOR operations

on integer representations of the inputs and returns the result in hexadecimal format.

**Algorithm 1** SAS-Based Device Linking Algorithm

**SD:** Generate DH pair $(d_{SD}, g^{d_{SD}})$ and random $R_{SD}$
Compute $C_{SD} = \text{commit}(R_{SD}, g^{d_{SD}})$
Send $C_{SD}$ and $g^{d_{SD}}$ to Server
**Server:** Forward $C_{SD}$ and $g^{d_{SD}}$ to PD
**PD:** Generate DH pair $(d_{PD}, g^{d_{PD}})$ and random $R_{PD}$
Compute $C_{PD} = \text{commit}(R_{PD}, g^{d_{PD}})$
Send $C_{PD}$ and $g^{d_{PD}}$ to Server
**Server:** Forward $C_{PD}$ and $g^{d_{PD}}$ to SD
**PD and SD:** Exchange $R_{PD}$ and $R_{SD}$ via Server
**PD:** Verify $C_{SD}$; **SD:** Verify $C_{PD}$
**PD:** Compute $K_{PD} = (g^{d_{SD}})^{d_{PD}}$
**SD:** Compute $K_{SD} = (g^{d_{PD}})^{d_{SD}}$
**Both:** Compute $SAS = \text{compute\_sas}(K_{PD}, R_{PD}, R_{SD})$
**SD:** Display $SAS$ in one of two ways:
**Option 1:** Display $SAS$ as a QR Code
**PD:** Scan and decode the QR Code to obtain $SAS$
**Option 2:** Display $SAS$ as a string
**PD User:** Manually enter the displayed $SAS$ on the PD
**PD:** Verify $SAS$
**if** $SAS$ matches **then**
    Send confirmation to SD
    **SD User:** Accept or reject the confirmation
    **if** confirmation accepted **then**
        Linking Device Mechanism Completed
    **else**
        Linking Device Mechanism Failed
    **end if**
**else**
    Linking Device Mechanism Failed
**end if**

**Algorithm 2** Compute Short Authentication String (SAS)

**Require:** $shared\_secret$, $random\_value\_A$, $random\_value\_B$
**Ensure:** Short Authentication String (SAS)
$sas\_int \leftarrow int\_from\_bytes(shared\_secret[:16], 'big')$
$sas\_int \leftarrow sas\_int \oplus int\_from\_bytes(random\_value\_A, 'big')$
$sas\_int \leftarrow sas\_int \oplus int\_from\_bytes(random\_value\_B, 'big')$
**return** $hex(sas\_int)[2:]$

## B Implementation Details

The implementation details are illustrated in Figure 12, provided in Appendix 12. This figure demonstrates how, in a typical E2EE messaging app's device linking mechanism, an attacker attempting to intercept the QR code and extract the SAS will fail to gain access to the user's WhatsApp account.

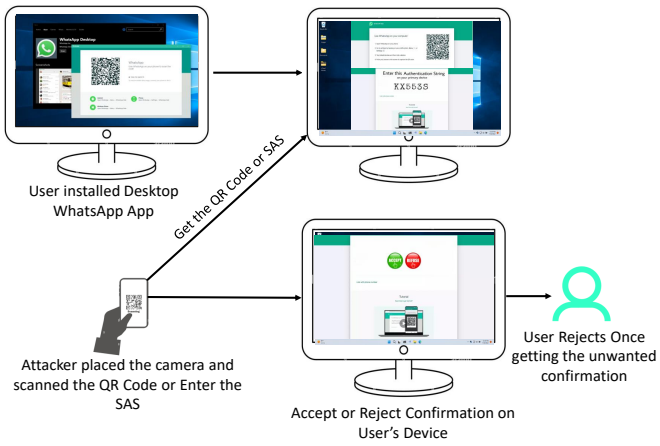


**Figure 12: SASLinker Implementation on E2EE Messaging Apps**