On Making U2F Protocol Leakage-Resilient via Re-keying

Donghoon Chang[1], Sweta Mishra[1], Somitra Kumar Sanadhya[2], Ajit Pratap Singh[1]

[1] IIIT Delhi, India. {donghoon,swetam,ajit1433}@iiitd.ac.in
[2] IIT Ropar, India. somitra@iitrpr.ac.in

Abstract. The Universal 2nd Factor (U2F) protocol is an open authentication standard to strengthen the two-factor authentication which is required to protect our authentication details online. It augments the existing password based infrastructure by using a specialized USB (*termed as U2F authenticator*) as the 2nd factor. In this work we show why the U2F protocol is not secure against side channel attacks.In U2F, at the time of manufacturing, the U2F authenticator is assigned fixed keys namely the device secret key and the attestation private key. These secret keys are later used by the U2F authenticator during the *Registration phase* to encrypt and provide digital signature to crucial informations that will help in proper validation of the user and the web server. However, the use of fixed keys for the above processing leaks information through side channel about both the secrets.
We present a countermeasure for side channel attack based on re-keying technique to prevent the repeated use of the device secret key for the encryption purposes and thus prevents side channel attack. We also recommend a modification in the existing U2F protocol to minimise the effect of signing with the fixed attestation private key. Incorporating our proposed countermeasure and recommended modification, we then present a new variant of the existing U2F protocol that we believe will provide strong security guarantees in the current existing scheme.
Keywords: Password, Authentication, U2F, Side-channel attack, Re-keying.

1 Introduction

Password based authentication is the most widely accepted and cost effective authentication technique in real world. In general practice, to ensure the confidentiality of a password, it is stored in a one-way transformed form by applying a technique known as 'Password Hashing'. Generally, the user selected passwords are easy to predict [1]. Therefore, it is easy for an attacker to create a dictionary of the most commonly used passwords and apply 'Dictionary attack' [2] to retrieve any password from its hash value. To prevent this offline dictionary attack, 'Password Hashing Competition' (PHC) [3] encouraged a resource consuming design for password hashing algorithms. The idea is to slow down the process of parallel computations by making computations resource intensive. To further enhance the security of passwords, use of cryptographic module for password hashing is suggested in [4]. However, these approaches are not sufficient to prevent online attacks, such as, phishing, Man-In-The-Middle (MITM), etc. Therefore the requirement of augmenting a second factor to strengthen the simple password based authentication is in recent trend. Few significant second factor solutions such as Time-based One-Time Password (TOTP) [5] and Short Message System (SMS) [6] are easy target for phishing attack or MITM attack which in turn affect user privacy severely and introduce major loss [7]. Frequent cases of phishing and monetary theft due to security flaws in password based solution have been reported. The U2F protocol proposed by Fast IDentity Online (FIDO) alliance in 2014 has been introduced as a strong augmentation that can overcome all known attacks currently faced in practice. Another protocol proposed by the FIDO alliance is called Universal Authentication Framework (UAF). The UAF protocol is an authentication protocol which supports biometric authentication to provide a unified and extensible authentication mechanism that supplants passwords [8]. As UAF supports biometric authentication, a different category of authentication when compared to U2F which supports passwords, in this work we mainly focus on the analysis of U2F protocol. The detailed analysis of UAF protocol is left out of the scope of this paper. The U2F solution is based on public-key solution [3]. Considering the claim of U2F developers that U2F is a significant contribution towards strengthening simple password-based authentication, a thorough third party analysis

[3] Public-key cryptosystem uses two mathematically related, but not identical, keys - a public key which is known to all and a private key which is known only to the owner. It accomplishes two functions: encryption and digital

of U2F solution is required to verify their assertion.

Motivation: In U2F protocol, we observe that at the time of manufacturing phase the U2F authenticator is assigned a unique device secret key and an attestation private key. Both these keys are fixed through out the life time of the token. The device secret key is used to compute a value called Keyhandle. The attestation private key is used to perform a signature. Both these operations are performed during the *Registration Phase* of the U2F protocol. Multiple execution of the *Registration Phase* involving computations with these fixed keys, leaks information through side channel about both the keys. In general, the U2F authenticator is a specialized USB and all crucial cryptographic operations are performed inside it. Physical access to the USB is not a difficult scenario and hence side channel attack is easy to mount. The side channel attacks are the most powerful and easy to implement attacks against cryptographic implementations [10,11]. Any cryptosystem is an easy target of this attack if suitable countermeasures are not adopted. The specifications [12,13] on U2F does not consider side channel in its security analysis. However, the security of the U2F solution depends on the keys which can easily be disclosed though side channel attack. Therefore, it is of utmost importance to protect U2F from this attack.

The side channel attacks (SCA), proposed by Kocher [10, 11], exploit weaknesses in the physical implementation of cryptosystem to recover the secret key information. These attacks treat ciphers as grey box and capitalize on the side channel leaks such as timing information [10], power consumption [11], electromagnetic leaks [14] etc. to correlate them with the internal states of the processing device which are dependent on the secret key. Several flavors of side channel attacks have been proposed depending upon the type of leakage investigated. Among them, power attacks are the most popular and extensively studied class of attacks. Power attacks analyze the power consumption of a cryptographic operation to recover the secret information. The 'Differential Power Ananlysis (DPA)' attack [11] is a more advanced form of power attack which allows an attacker to compute the intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations [10]. DPA attack is very practical to implement. When a cryptographic operation is performed over a fixed key, computation with different input-output pairs can easily reveal the key by applying DPA attack. Template attack [15] is another powerful side channel attack which can reveal the secret with less number of input-output pairs but not practical till date. It requires significant pre-processing to be implemented but can break implementations secure against DPA attack as well. As template attack is far from being practical, having a solution that is DPA resistant is enough to claim side channel resistance. Therefore, nowadays it is essential to consider the implementations of cryptographic algorithms to be side channel resistant. As observed, the use of fixed keys in U2F protocol can similarly leak information about both the secrets applying the DPA attack. Therefore, we explore the possible cryptographic solutions to prevent repeated use of the fixed keys for U2F protocol. We show the rekeying technique which generates different session keys, as a suitable countermeasure.

In cryptography, rekeying is a technique where different subkeys are derived from a master key to limit the number of operations performed under the same master key [16]. It is considered as an efficient approach to prevent leakage of a secret key information through side channel by generating multiple different keys from the initial secret key and limiting the number of operations performed with each generated keys. In our proposed solution, we utilize rekeying technique to protect the computations that are performed under fixed keys in the U2F protocol.

Our Contribution:

1. We first show why the U2F protocol cannot be secure against side channel attacks. We then propose a countermeasure to fix this issue. Our proposed solution is based on the rekeying technique that derives session keys from the device secret key. Use of different session keys instead of the fixed device secret key helps in keeping the device secret key confidential.
2. The signature function that uses the attestation private key also leaks information that can be captured through side channel analysis. Usually in public key infrastructure (PKI) [9], the verification of digital

signature. For encryption a sender encrypts with the public key of the receiver so that only the receiver can decrypt with its own private key. For digital signature the sender signs a known message with its private key so that the receiver can verify with the public key of the sender. For more details one can refer to [9]

signature is satisfied by the use of certificate. Intuitively, it may seem that use of session key will prevent the side channel attack on attestation private key as well. However, getting valid certificate for each session key generated from attestation private key is not practical in asymmetric cryptosystem. Hence, it is difficult to prevent side channel attacks using the existing solutions for signature. Therefore, we recommend a modification to mitigate the attack.

3. Incorporating our proposed countermeasure and recommended modification, we then present a new variant of the existing U2F protocol that we believe will provide strong security guarantees in the current existing scheme.
4. There is a lack of clear and comprehensive literature that describes the U2F protocol with detailed security analysis. We present a detailed analysis of the U2F protocol including its security analysis. We also try to fill the gap in the description of the protocol by providing clear explanation of all the cryptographic operations.
5. We also explain in brief how the side channel attacks of U2F protocol and the corresponding proposed countermeasures are similarly applicable to Universal Authentication Framework (UAF) protocol.

The rest of the paper is organized as follows. In Section 2, we present an overview of the U2F protocol. In Section 3, we detail the existing security analysis on U2F protocol. Possible side channel attack points that can be exploited in U2F protocol are explained in Section 4. Subsequently, the countermeasure on these attacks and the complete description of our modified U2F protocol incorporating the countermeasure are documented in Section 5. This is followed by a detailed explanation of the design rationale behind our proposed modifications and its security evaluation in Sections 6 and 7 respectively. In Section 8, we present a discussion on the few limitations of the U2F protocol. The Section 9 explains in brief how the side channel attacks of U2F and the corresponding countermeasures can be applied to UAF protocol as well. Finally, in Sections 10, we conclude our work.

2 Overview of the U2F Protocol

The U2F protocol is considered over a password-based authentication system where a web server allows n users identified with their usernames such as $u_1, u_2, \ldots, u_n$, and their corresponding passwords $p_1, p_2, \ldots, p_n$. For instance, the u_i and p_i denote the username and the password of the ith user. The server maintains a database listing pairs of username and corresponding hashed password as follows.

$$(u_i, H(p_i \parallel s_i))$$

where H is the password hashing algorithm imposed by the server and s_i is the server generated salt corresponding to each user u_i which is a fixed length random value. U2F is a second factor authentication protocol that augments this simple password-based authentication system and is explained in the subsequent subsections.

2.1 Notations

The key notations used in this work are provided in Table 1.

2.2 Universal 2nd Factor (U2F) Protocol

It is a protocol proposed by Fast IDentity Online (FIDO) alliance in 2014 as U2F v1.0 [12] and later in 2016 as U2F v1.1 [13]. The protocol description is similar for both the versions. It allows online services to enhance the security of the existing password infrastructure by adding a strong second factor called U2F at the time of user login. The augmentation is claimed to be a secure and user friendly solution. To use U2F, the user needs to login with a username and password and then perform a simple button-press on a USB device (in case of U2F being a USB). A single U2F authenticator can be used across all online services that support the protocol with built-in support in web browsers of the client.

The U2F protocol allows the U2F authenticator to be either a secure hardware or a software. In the following description we use a generic term U2F token to express both hardware or software. The U2F

Table 1: Notations

Relying Party/Origin	Web server or Server
FIDO Client	Web browser of the client
FIDO Authenticator	U2F token (hardware/software)
AppId	URL of the web server supporting U2F
DS_K	Unique secret key assigned to the U2F token
K_i	i^{th} session key generated from DS_K and the counter i
PK_u	Public key of the AppId corresponding username u
SK_u	Private key of the AppId corresponding username u
PK_M	Attestation public key of the manufacturer
SK_M	Attestation private key of the manufacturer
PK_T	Public key of the U2F token
SK_T	Private key of the U2F token
$Sign_x(m)$	Signature of message m with key x
ACert	Attestation certificate issued by a trusted CA to the manufacturer
$\|\|$	concatenation operator
$\mathcal{K}(\cdot)$	a random asymmetric keypair generation function
s	salt, a fixed length public random value
$f^{(i)}(K_0)$	i times recursive call of function f starting from initial value K_0
$x \xleftarrow{\$} \{0,1\}^n$	randomly generated n-bit value x
$DB \xleftarrow{add} x$	adding the value x to the database DB
R	A r-bit challenge randomly generated by the server
Kh	A key container called Keyhandle generated by the U2F token at the time of 'Registration' which contains AppId and private key SK_u
$H(x)$	Computing hash on the value x where H is any cryptographic hash function
CA	Trusted Certificate Authority under PKI infrastructure
S_db	A database maintained by the server to keep records
U2F_db	A database maintained by the U2F token to keep records

protocol supports two generic steps for online authentication, namely, 'Registration' and 'Authentication'. Both of these processes involve three parties: U2F token (hardware/software) which is also referred to as 'FIDO Authenticator', the web browser of the client which is called 'FIDO client' and the web server which is referred as 'Relying Party'. The U2F token is a crucial part for the analysis of the protocol. Therefore, we explain the protocol in three steps, appending the process of assigning the secret keys to the U2F token by the manufacturer at the time of manufacturing as the first step. Following is the description of these three steps of the protocol.

2.2.1 U2F Manufacturing Phase This phase is executed at the time of manufacturing of the U2F token as shown in Fig 1. In this phase the U2F token is provided with a randomly generated unique secret called Device Secret Key DS_K at Step1. At Step2, a public-private key pair (PK_M, SK_M) is provided to the token. The (PK_M, SK_M) keypair is not a one-time generation. Once randomly generated from a key generation function, the manufacturer provides the same (PK_M, SK_M) pair to multiple tokens. The scenario can be interpreted as the keypair (PK_M, SK_M) generated for the manufacturer and shared with all the tokens manufactured by it. At Step3, a certificate Acert issued by a trusted CA which includes PK_M (the public key of the manufacturer) is provided to the token. During the registration phase, the signature with SK_M is verified with PK_M which is extracted from Acert by the server. This verification proves the genuineness of the U2F token. Therefore, both DS_K and (PK_M, SK_M) are fixed for a token while the same (PK_M, SK_M) values are provided to multiple tokens which preserve the anonymity of the token. An integer counter 'ctr' is provided at Step4. This 'ctr' is initialized to value zero and gets incremented after each successful execution of the authentication phase as explained in Section 2.2.3.

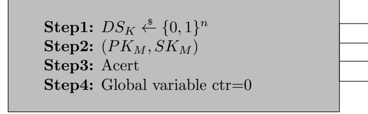


Fig. 1: **U2F Manufacturing Phase.** The key DS_K is a randomly generated n-bit secret. The keypair (PK_M, SK_M) is the asymmetric keypair of the manufacturer shared with the token. Acert is the certificate signed by a trusted CA which includes PK_M. At the time of registration phase, a signature with SK_M is verified with PK_M from Acert by the server to prove the genuineness of the U2F token. The global variable ctr is initialized to zero and incremented after each successful authentication as explained in Section 2.2.3

2.2.2 U2F Registration Phase The registration procedure is explained in Fig 2. Initially the username u and password p are communicated to the server at Step1. At Step2 the server verifies the values u and $H(p \parallel s)$ which is the hash of the password where s is the salt. On successful verification, the server generates a r bit random challenge R to differentiate each request, else it rejects the registration request. The server sends its identity as AppId and the value R to the browser at Step3. At Step4 the values are forwarded to U2F token along with the (TLS) channelId of the browser for the server. The TLS ChannelId is a TLS extension, originally proposed in [17] as Origin-Bound Certificates and its refined version is available at IETF Internet-Draft [18]. Receiving the information, at Step5, the U2F token first generates a website specific public-private key pair represented as (PK_u, SK_u) by applying a random key generation function $\mathcal{K}(\cdot)$. The function $\mathcal{K}(\cdot)$ can be an openSSL key pair generation library which needs as input an elliptic curve such as the NIST standard P-256 curve. We define a key dependent function $f_K(\cdot)$ where K is the secret key. The U2F token computes the function with key DS_K and outputs f_{DS_K} (AppId $\parallel SK_u$) which is called the Keyhandle Kh. The above function can be instantiated with any block cipher such as AES in CBC mode or HMAC as shown in Fig. 3 and 4 respectively. The values (PK_u, SK_u) and Kh can optionally be stored in a database denoted as U2F_db inside the U2F token. It then computes a signature $\mathcal{S} \leftarrow \text{Sign}_{SK_M}$(AppId, R, ChannelId, PK_u, Kh). At Step6, the token sends the values PK_u, Kh and $\mathcal{S}$ along with ACert to the browser. The browser forwards the received values to the server at Step7. After receiving the values, the server first verifies the signature $\mathcal{S}$ with PK_M from the certificate Acert. This verification proves the genuineness of the U2F token. On verification, the server updates its database S_db with the values (PK_u, Kh, Acert) corresponding to the username u and it shows the successful registration of the U2F token else the server rejects the registration request. The U2F protocol allows a user to register multiple tokens with the same account.

2.2.3 U2F Authentication Phase The authentication procedure is explained in Fig 5. Once second factor authentication with U2F is registered, for example, when the values $(u, H(p \mid\mid s), PK_u, Kh,$ Acert) are registered with the server for a particular user u, the subsequent login to the website needs to verify the registered values by communicating with the U2F token through browser. The steps are as follows. After receiving username u and password p at Step1, the server checks its database to retrieve the Keyhandle Kh for u at Step2. On successful verification, the server generates a r bit random challenge R and sends it along with Kh and its AppId to the browser at Step3. The browser forwards the received information with ChannelId to the U2F token at Step4. At Step5, the U2F token first verifies the received values by performing inverse computation ${f_{DS_K}}^{-1}(Kh) = (\text{AppId}' \mid\mid SK'_u)$. It then compares the received AppId$'$ with the stored AppId. Similar checks can be done if $f_K(\cdot)$ is a keyed hash as shown in Fig. 4. If a successful match happens, the token increments a counte represented as 'ctr'. This 'ctr' can be a global or a local variable. If it is local variable then each AppId gets its own 'ctr' else a single 'ctr' is used across all registered AppId. Throughout the explanation we consider 'ctr' as a global integer variable. The value of the 'ctr' is incremented after each successful authentication by the U2F token. This value 'ctr' is introduced to detect cloning of the U2F token as explained in Section 6.2. The U2F token signs the received values of step4 and 'ctr' with SK_u which is represented as $\mathcal{S}$ and sends the values $\mathcal{S}$ and ctr to the browser at Step6. The browser forwards the values to the server at Step7. Finally the server verifies the received signature $\mathcal{S}$ with stored key PK_u for the username u from S_db. On successful verification, it keeps the 'ctr' value with the database and this shows the successful completion of the authentication process.

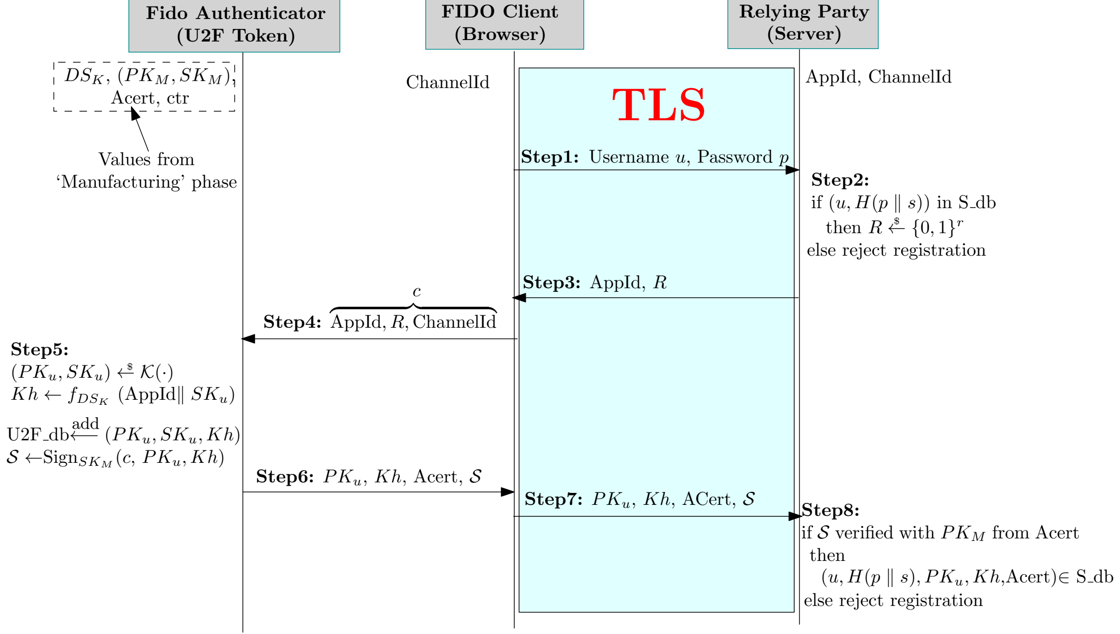
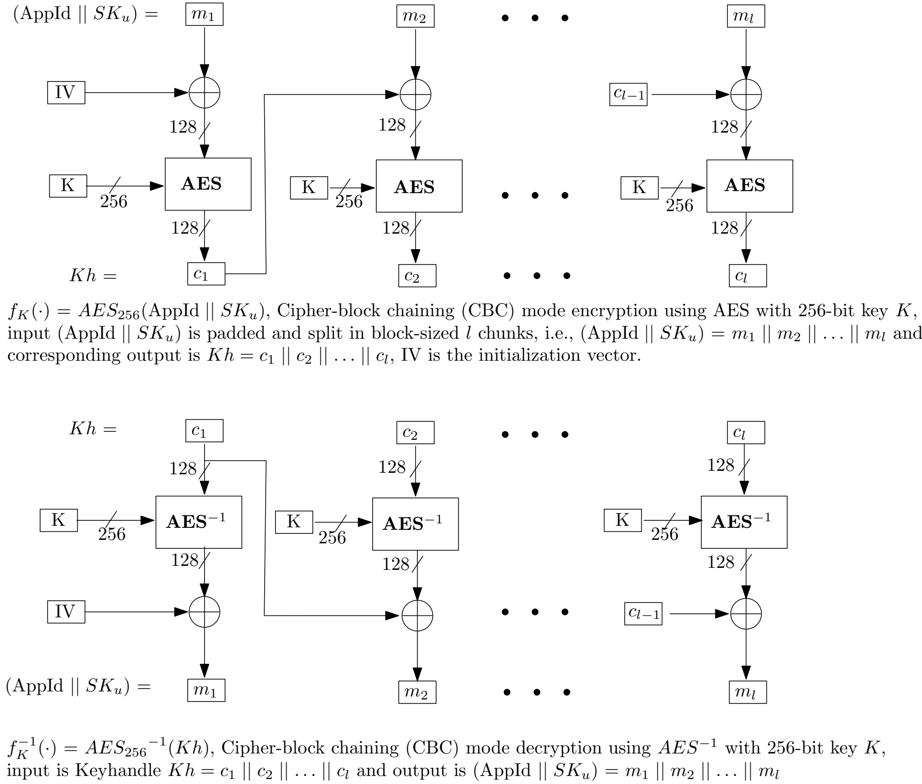


Fig. 2: **U2F Registration Phase.** The key DS_K, the asymmetric keypair(PK_M, SK_M) with attestation certificate Acert which includes PK_M and the counter 'ctr' are from the manufacturing phase. AppId is the URL of the server. ChannelId is the TLS ChannelId. S_db is the server database. R is a r bit random number generated by server. $\mathcal{K}(\cdot)$ is a random key generation function which generates the keypair (PK_u, SK_u). $f_{DS_K}(\cdot)$ with key DS_K is the function to generate Keyhandle Kh as shown in Fig. 3 and 4, U2F_db is the database at U2F token.



$f_K(\cdot) = AES_{256}$(AppId $\parallel$ SK_u), Cipher-block chaining (CBC) mode encryption using AES with 256-bit key K, input (AppId $\parallel$ SK_u) is padded and split in block-sized l chunks, i.e., (AppId $\parallel$ SK_u) = $m_1 \parallel m_2 \parallel \ldots \parallel m_l$ and corresponding output is $Kh = c_1 \parallel c_2 \parallel \ldots \parallel c_l$, IV is the initialization vector.

$f_K^{-1}(\cdot) = {AES_{256}}^{-1}(Kh)$, Cipher-block chaining (CBC) mode decryption using AES^{-1} with 256-bit key K, input is Keyhandle $Kh = c_1 \parallel c_2 \parallel \ldots \parallel c_l$ and output is (AppId $\parallel$ SK_u) = $m_1 \parallel m_2 \parallel \ldots \parallel m_l$

Fig. 3: Instantiation of $f_K(\cdot)$, ${f_K}^{-1}(\cdot)$.

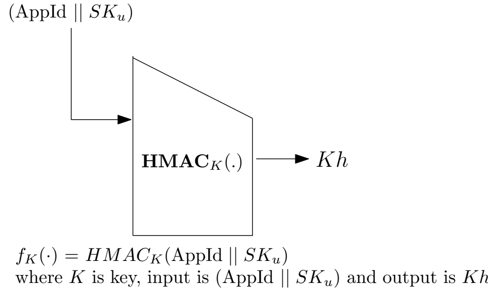


Fig. 4: Instantiation of $f_K(\cdot)$.

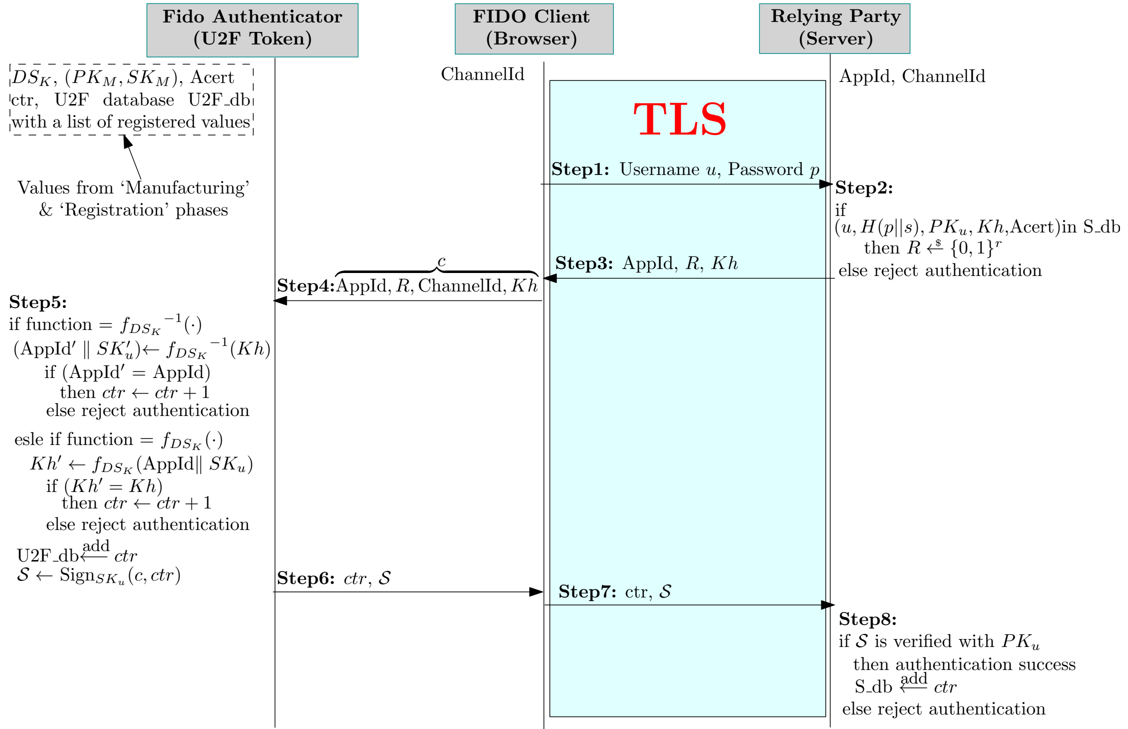


Fig. 5: **U2F Authentication Phase.** The key DS_K, the asymmetric keypair(PK_M, SK_M) with attestation certificate Acert which includes PK_M and counter 'ctr' are provided at the manufacturing phase. The U2F_db contains all the registered values from registration phase. The server database S_db contains all the required values generated at the registration phase corresponding to a registered username. R is a r bit random number generated by the server. AppId is the URL of the server. ChannelId is the TLS ChannelId. The value Kh is the Keyhandle. The verification of received Kh for both the cases of Step5 is shown in Fig. 3 and 4. 'ctr' is the counter used to record the number of authentication requests successfully satisfied by the U2F token.

2.3 U2F v1.0 [12] and U2F v1.1 [13]:

Considering the specifications provided for U2F v1.0 [12] and U2F v1.1 [13], the difference between both the versions are in the optimization of the implementation, not at the protocol level. Following are the listed differences as documented in the specifications for both the versions.

- U2F protocol allows multiple tokens to be registered under same AppId for a specific user. Therefore, for each token, different Keyhandles are generated for a specific AppId. In U2F v1.0, a separate AppId and challenge pairs were sent for every keyHandle, whereas U2F v1.1 suggests an optimization over it. The U2F v1.1 allows a single AppId and challenge pair for multiple keyHandles registered for the AppId.
- The U2F v1.1 JavaScript API specification supersedes JavaScript API of U2F v1.0. The major difference between these two versions is the way the requests to be signed are formatted between the relying party and the client.

3 Existing Security Analysis of U2F Protocol

This section presents the current security analysis of U2F protocol.

3.1 Denial of Service (DoS) Attack

Denial-of-service attacks are characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service [19].

Following the explanation of the protocol in Section 2.2, the server needs to maintain a user database that stores user specific Keyhandle and public-key. Any modification of these values by an attacker may result in DoS attack for a legitimate user. For example, at server side an attacker can modify the public key from the database [20]. Due to this modification, access will be denied to the legitimate user after verification of Step8 of Fig. 11. Even though the public key in asymmetric setting is supposed to be a public value, in view of the attack this public key needs a secure storage.

At client side an attacker with access to a U2F token may register U2F authentication for an account of a third party without his notice. This is possible with the assumption that the username and password of the account holder is known to the attacker. This way the legitimate user would be prevented to access his account for not having the registered token. This attack is difficult to prevent as it needs the passwords to be difficult to guess and known only to its user.

Another possibility of DoS attack is when the attacker gets access to the U2F token of a legitimate user. Suppose the token has limited memory and the attacker has registered multiple websites of its own choice, leaving no storage for any other request. As there is no way to know the list of websites registered under a U2F token by its user, it is difficult for the legitimate user to notice this attack. The user can only receive denial for new registration. It shows keeping the U2F token safe is mandatory to resist this DoS attack.

3.2 Parallel Signing Request attack

The scenario of this attack involves two U2F enabled web pages, loaded from different websites [7]. It assumes that the attacker can detect the web pages loaded by the user browser so that it can synchronize the two U2F operations initiated for two different websites by the user. As per the specification of U2F v1.0, there is no way to notify the user about which operation is performed by the U2F token. User performs a button-touch for both ‘Registration’ and ‘Authentication’ process as only the user consent is mandatory for the execution of both the operations. Therefore, the use of the U2F token for monetary transactions could be at high risk. Let, an attacker can observe and control all activities in the current browser of a user and the user be authenticated to a banking website. Then the attacker can force the user to confirm a transaction with the opened banking website which is initiated by the attacker. This is possible when two parallel websites are operated with U2F support in the user browser and both asking for the user consent to initiate U2F authentication. In real scenario, U2F is a specialized USB which triggers authentication request multiple times due to some connection issue or after expiration of each session apart from its general functionality.

Therefore, it is difficult to know for the user the real cause of authentication request by the U2F token. Suppose one website is requesting for U2F authentication which is visible at the foreground and at the same time a transaction page (initiated by the attacker) is working at the background opened as an iframe. User provides his consent for the displayed website while the first request that reached to U2F was for confirming the transaction. This shows a successful execution of the transaction without the knowledge of the user. Therefore the second factor authentication with U2F is not safe for transactions.

FIDO alliance, in their specification has already mentioned that U2F never claims transaction non-repudiation. Therefore, it is not an attack for U2F. However, existing second factor authentications, such as SMS or OTP provide this assurance. Even the version U2F v1.1 recommends to have notification for each operation but it is not mandatory. Therefore, with existing specifications, we can consider U2F as a general solution to secure the password-based authentication. Augmenting OTP and U2F together can protect the transaction, however it adds overhead and contradicts one of the goals of U2F to be user-friendly.

With growing popularity of Internet banking, cases of online frauds have also increased manifold, resulting in considerable financial losses [21]. The banking frauds have increased 93% in 2009-2010 [22], and 30% in 2012-2013 [23] and it is ever growing. Internet banking or online transaction frauds are difficult to analyze and detect because every day we come across a new attack technique and detection by the user is usually delayed [21]. Therefore, claiming transaction-nonrepudiation[4] is an essential property for strong and secure authentication scheme. The online users are always concerned about the safety of their financial property and therefore every new second factor proposal must provide higher security than the existing ones to be considered a strong solution. As U2F lacks transaction non-repudiation, it is only a general solution to protect the simple password-based authentication.

3.3 Phishing attack

The concept of the phishing attack was first introduced in 1987 at Interex conference by Jerry Felix and Chris Hauck [25,26]. NIST in [27] defines phishing as 'a technique which refers to use of deceptive computer-based means to trick individuals into disclosing sensitive personal information'. Commonly, it is a form of social engineering that uses email or malicious websites to solicit personal information from an individual or company by posing as a trustworthy organization or entity.

U2F claims to be phishing resistant solution and two different scenarios prove the claim by showing how phishing can easily be detected at the time of authentication phase of the U2F protocol [7]. In one scenario, when the U2F is registered with a genuine website (AppId) and at authentication phase if a phishing website (AppId′) forwards the Keyhandle and challenge from that genuine website (corresponding AppId), the U2F can easily detect the phishing attempt with origin mismatch. Following Fig. 5, the scenario can be described as the U2F receives AppId′ at Step4 and the Keyhandle provides AppId at Step5 and hence both AppIds differ in value. Hence phishing is detected at the U2F token. In the second scenario, if an attacker forwards challenge (R) from a genuine website and Keyhandle (Kh') from a phishing website, U2F token signs the challenge with the corresponding private key of the received Keyhandle. If the signature is forwarded to genuine website, it will reject as the signature would not be verified (Step8 of Fig. 5). Therefore, because of the signature and origin specific Keyhandle, existing approaches for phishing attack fails to be applied on U2F protocol.

3.4 Man-In-The-Middle (MITM) attack

In the MITM attack, the common scenario involves two legitimate endpoints (victims), and a third party (attacker) [28]. The attacker has access to the communication channel between two endpoints, and can eavesdrop, insert, modify or simply replay their messages. As mentioned in [28], the term Man-In-The-Middle attack was first mentioned by Bellovin et al. in [29].

The U2F specifications [12,13] show how this technique prevents MITM attack where the MITM attacker intermediates between the user and the webserver. As mentioned in the U2F specifications [12, 13], MITM attack is possible when the following two conditions are satisfied.

1. MITM attacker is able to get a server certificate for the origin name issued by a trusted CA, and

[4] Nonrepudiation is the assurance that someone cannot deny something [24].

2. Channel IDs are not supported by the browser.

However MITM attack is possible contradicting the developers claim. An MITM attack is shown on TLS with Channel ID support which is based on following conditions [30].

1. MITM attacker is able to get a server certificate for the origin name issued by a trusted CA, and
2. Channel ID is supported by the browser.

The attack is similar to the attack in [31] which is also applicable to the U2F based authentication and called 'Man-In-The-Middle-Script-In-The-Browser (MITM-SITB)' [30]. The MITM-SITB attack works as follows. It assumes that TLS Channel ID is supported by the browser and the communicated webserver possesses valid/invalid certificate as it works with invalid certificates as well. This is because, in real scenario, the web browser only sends warning of invalid certificate to the user and proceeds according to the user action. In most of the cases user ignores the warning. Hence, invalid certificate also helps the attacker to launch the attack. Let the attacker knowing the website request of the user in advance, compromises the DNS server to divert the route of that legitimate website to a malicious website which contains same origin address but hosted in some different server. Next, when the user initiates authentication to the compromised website using U2F, it establishes a TLS connection with the malicious website. The malicious website pushes a malicious JavaScript code to the browser, without the notice of the user browser, and terminates the connection. The browser re-establishes a fresh TLS connection but with the legitimate website (as attacker places the legitimate IP at DNS server) for subsequent communication with it. The attacker becomes passive after injecting the code and the user authentication is performed over the connection between the browser and the legitimate website. Browser supports 'Single Origin Policy (SOP)' which means webpages can interact with each other only if it belongs to same origin. Therefore the malicious code which also includes the same origin of the legitimate website, gets access to the legitimate webpages. It is difficult to detect the attack as the TLS channel ID will be same for both the browser and the server. U2F only verifies the Channel ID of the browser which is origin-specific, and hence can not detect the attack.

Real world attacks are multi faceted. While all the above attacks are based on the common online attacks of Internet and do not require tampering of the U2F token or exploiting the U2F protocol, there are other class of attacks which can exploit the U2F token/protocol. One such class of attack is side channel attack. The attack explained in the following section is the attack on the U2F protocol based on the powerful side channel analysis. When a cryptographic device can get in the hands of the adversary or otherwise become easily accessible then one needs to pay close attention to side-channel attacks. The U2F solution is based on a hardware token (in common practice) and hence can easily be accessible to the attacker. As side channel attacks are easy to implement and the most successful cryptographic attacks to compromise secrets, it is of utmost improtance to protect U2F from side channel attack.

4 Possible Side Channel Attacks on U2F Protocol

The side channel attack is one of the most powerful attacks since the seminal work of 1996 by Paul Kocher [10]. It is an attack based on information gained from the physical implementation of a cryptosystem [32]. There exists different forms of leakage under side channel. For example, timing information [10], power consumption [11], electromagnetic leaks [14] etc., which provide information that can be exploited to break the system. Therefore, nowadays it is essential to consider the implementations of cryptographic algorithms to be side channel resistant. For some specific cryptographic operations, countermeasures are available. As side channel attacks are easily implementable, current cryptographic proposal tries to incorporate leakage resilient implementation considering coutermeasures. Kocher's first work was based on timing attack [10] and other significant contribution was 'Differential Power Ananlysis (DPA)' attack [11]. DPA is a more advanced form of power analysis which allows an attacker to compute the intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations [10]. DPA attack is very practical to implement. When a cryptographic operation is performed over a fixed key, computation with different input-output pairs can easily reveal the key by applying DPA attack. In the following section our proposed attack is based on the observation that different input-output pairs in U2F protocols are computed over a fixed key. Therefore, we claim DPA attack is possible on the

implementation of U2F protocol. The 'Template attack' [15] is another powerful side channel attack which can break implementations secure against DPA attack but it is far from being practical. The template attack requires significant pre-processing to be implemented. Specifically, it requires a great number of traces to be preprocessed before the attack can begin. Therefore, considering the current state-of-the-art, having a solution for a cryptographic implementation that is DPA resistant, is enough to claim that it is side channel resistant.

In the subsequent subsections we explain how side channel attack can be launched on the U2F protocol.

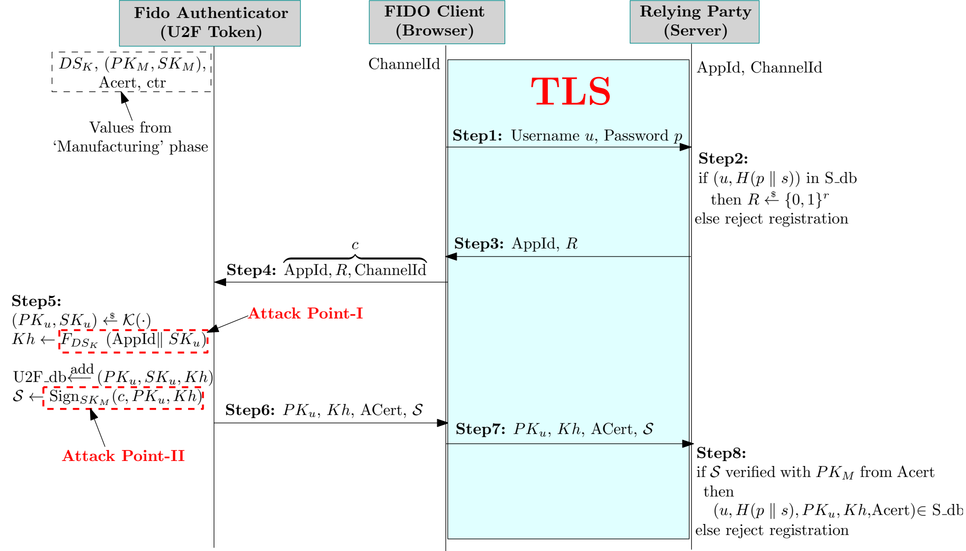


Fig. 6: **Side channel attack points at Registration Phase.** The key DS_K, the asymmetric keypair(PK_M, SK_M) with attestation certificate Acert which includes PK_M and the counter 'ctr' are from the manufacturing phase. AppId is the URL of the server. ChannelId is the TLS ChannelId. S_db is the server database. R is a r bit random number generated by server. $\mathcal{K}(.)$ is a random key generation function which generates the keypair (PK_u, SK_u). The function $f_{DS_K}(.)$ with key DS_K generates Keyhandle Kh as shown in Fig. 3 and 4. U2F_db is the database at U2F token. The Attack Point-I is to compromise the key DS_K and the Attack Ponit-II is to compromise the key SK_M.

4.1 Side Channel Key Recovery attack on Device Secret Key DS_K

According to the specifications [12,13], all U2F tokens contain a unique device secret key DS_K which is used to generate Keyhandle Kh as shown in Fig. 3 and 4. Specifically, at the time of registration, the U2F token computes $Kh \leftarrow f_{DS_K}(\text{AppId} \parallel SK_u)$ where AppId is the URL of the server and SK_u is the private key for the username u corresponding to the AppId. With the access to the U2F token, an attacker can collect multiple power traces for different input-output pairs operated under the function $f_{DS_K}(\cdot)$ with fixed key DS_K. This is possible when the attacker gets the U2F token and requests registration for different AppIds for U2F authentication. We assume that the communication between the browser and U2F token is easy to capture by the attacker. This assumption is based on the fact that the information received or sent by the browser can be easily captured through the Document Object Model (DOM) tree created by the browser as explained in [33]. Therefore attacker knows the different inputs (AppId) and the corresponding outputs (Keyhandle) which again can be obtained through intercepting the communication of the U2F token and browser and applies DPA to get DS_K from $f_{DS_K}(\cdot)$. The point of the attack at the time of registration with U2F protocol is shown in Fig. 6 as 'Attack point-I'. Through side channel, it is possible to compromise the key DS_K from simple differential power analysis as explained above. To prevent this attack, repeated use of fixed DS_K for different input-output (AppId-Keyhandle) pairs should be prevented. Use of session key is one solution and our proposed countermeasure is based on this concept.

4.2 Side Channel Key Recovery Attack On Attestation Private Key SK_M

As explained in Section 9.1, the attestation private key SK_M along with the certificate Acert which contains PK_M signed by some trusted CA are provided by the manufacturer to the U2F token at the time of manufacturing. This SK_M is used to sign at the time of registration to prove genuineness of the U2F token to the server. Same attestation key is provided by the manufacturer to a large number of tokens generated at some specific time period to preserve anonymity. The signature with SK_M at the time of registration leaks information of the key SK_M through side channel such as DPA attack. The attack point is shown in Fig. 6 as 'Attack Point-II' which is the signature Sign_{SK_M}(AppId, R, ChannelId, PK_u, Kh). Therefore, an attacker with access to the U2F token can collect multiple traces for different input (containing different values of AppId, R, ChannelId, PK_u, Kh) and output (signature) pairs signed under SK_M as every signature at the time of registration is performed with the fixed key SK_M. To preserve anonymity, a large number of devices are using this fixed SK_M. Therefore, compromising SK_M targeting a single U2F token, in practice compromises a large number of tokens which contains the same SK_M. Obtaining SK_M, an attacker can simulate the U2F token in a software programme and use it to prove the possession of the token. It is almost impossible to detect which device has been compromised and therefore requires to invalidate all tokens with compromised SK_M. Use of different public-private key pair for attestation can prevent this mass compromise.

There are two approaches to show a system is side channel resistant. One by having a hardware preventing attack and the other by providing an algorithmic solution. In the following section we present the algorithmic solution to prevent/mitigate the side channel attack on U2F protocol.

5 Proposed Countermeasures

In this section we explain a coutermeasure to prevent the recovery of the device secret key from side channel attack. Specifically it is a counter-based re-keying process, a concept which is first analysed in [16] and then an efficient tree-based approach is explained in [34]. We suggest modification in the approach of assigning attestation private key to U2F tokens to mitigate the attack mentioned in Section 4 which recovers the key SK_M.

5.1 Countermeasure to protect Device Secret Key

Our proposed side channel resistant solution to protect the device secret key DS_K generates session keys sequentially from DS_K which is a counter based re-keying technique explained in Algorithm 1. The function g of the proposed algorithm takes the previous session key and current counter as input and generates the current session key. The use of counter for re-keying is a natural way to avoid repetition of keys. However, it has various drawbacks depending on the setting and application. In case of client-sever model which is the most common setting, it suffers from synchronization issue when session key is generated by the client and the corresponding counter is communicated to the server. This is required with the assumption that initial secret is known to both the client and the server. Considering the U2F protocol where sharing of keys with the server is not required, the significant problem is the computation overhead inside the U2F token. The overhead increases with the value of the counter and we address how to balance the overhead as well at the following subsections.

The common approaches to derive session keys are based on either a parallel approach where $K_i = g(K, i)$ when K is the initial secret and i is the counter, or sequential, using the previous session key to derive the current one[16]. The sequential approach is better to avoid DPA as it changes the key at each execution depending on the previously generated key unlike the case of parallel approach where every execution depends on the initial secret. The sequential approach also provides forward secrecy i.e., disclosure of the current key does not reveal the previously generated session keys. However, it has a problem with efficiency. The value of the counter reveals the number of computations required to derive the session key starting from the initial secret. If there is a significant distance between the required key and the initial secret then huge number of computations are needed to reach the desired value in case previous values are not stored. For example, to compute the ith session key, it requires a single computation if the $(i-1)^{th}$ session key is stored. Now consider a case where it is required to re-generate j^{th} session key, $j << i$ and

the ith one is stored. In such case we need j computations starting from the initial secret to re-derive the jth session key which causes the overhead. The scenario of computation overhead as discussed in the example is equally applicable to our proposed techniques. However, we provide the way to balance the computation overhead which is stated under the explanation of usability of each techniques mentioned below.

Algorithm 1 To generate session key from device secret key
Input: AppID, *Counter*, Device secret key DS_k, Previous session key, U2F database U2F_db
Output: A tuple (Keyhandle, *Counter*)

1: Global var *Counter* initialized at 1;
2: $K_0 \leftarrow DS_K$
3: Set i = *Counter*
4: $(PK_u, SK_u) \xleftarrow{\$} \mathcal{K}(.)$
5: $K_i \leftarrow g(K_{i-1}, i)$
6: $Kh_i \leftarrow f_{K_i}(\text{AppId} \mid\mid SK_u)$
7: U2F_db $\xleftarrow{add} (Kh_i, i)$
8: *Counter* = *Counter* +1
9: return (Kh_i, i)

We provide some practical approaches to instantiate the function g of Algorithm 1. We represent the function g as g(key, counter) where key is an n-bit secret value and *Counter* is an integer.

1. **Proposed Technique 1:** In this case g(key, *Counter*) = $HMAC_{key}(Counter)$. The function g is a HMAC with any cryptographically secure hash function. The key is the previous n-bit session key K_{i-1} and input is the current m-bit counter i which produces the next n-bit session key $HMAC_{K_{i-1}}(i)$ as shown in Fig. 7. Initially a system-wide global variable *Counter* i is initialized at value one. It gets incremented at each registration request satisfied by the U2F token. The *Counter* is handled internally by the token therefore no external control is possible. As the *Counter* i is incremented for each new registration, we get different session keys K_i. This K_i is then used for computing Keyhandle $Kh_i \leftarrow f_{K_i}(\text{AppId} \parallel SK_u)$. Therefore, the value i is associated with the value Kh_i and the pair (Kh_i, i) is stored internally inside a database of U2F token represented as U2F_db. At the time of authentication, the value of Kh_i is verified only if the correct value of i is provided.

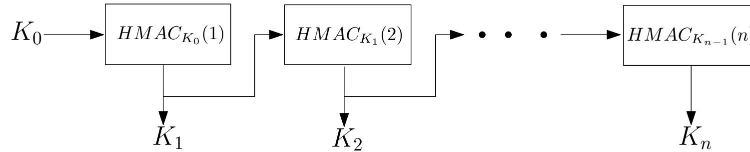


Fig. 7: **The function** $g(K_{i-1}, i) = HMAC_{K_{i-1}}(i) = K_i$ where $K_0 = DS_K$. It generates n session keys sequentially.

.

Security: The function $HMAC$(key, *Counter*) works as a pseudorandom number generator [5]. Therefore, it increases the lifetime of the device-secret key and achieves the provable security gains in practice [16]. This sequential approach of key generation also provides forward secrecy, i.e., disclosure of the current key does not reveal the previously generated session keys. To protect future keys, use of random nonce instead of *Counter* is advisable. In that case nonce value should not be shared outside the U2F token.

[5] In cryptography a pseudorandom number generator is an algorithm which generates a sequence of numbers that are computationally indistinguishable from true random numbers [35].

Then the keyhandle could be used as the index of the database inside the U2F token.

Usability: Our aim is to achieve DPA resistance and at the same time efficient computation inside the U2F token. This *Counter* based approach is efficient in terms of avoiding key repetition, but has the problem with efficiency. When the current *Counter* is large enough from the initial value, it increases the required number of computations significantly if only the initial secret is available.

The computation of every new session key needs one extra hash computation if the previous session key is stored. Therefore, at the time of registration, overall the overhead is of one hash computation with existing process. In continuation of the example of Section 5.1, suppose the current *Counter*=i and corresponding session key K_i is stored at the U2F token. At that time authentication request for j^{th} registered value where $j << i$ reaches to the token. As $(j-1)^{th}$ session key is not stored, it requires j computations to verify the values. Therefore, at the time of authentication, if the current counter value is significantly large, the computation overhead is also significant to re-derive the corresponding session key if the computation starts with the initial secret. We can balance the overhead of authentication phase. One possible solution is to first consider the speed of single HMAC computation and then to measure the feasible number of HMAC computations that can be performed in allowable time. Then store the session key values maintaining a fixed gap (depending on the allowable HMAC computation) covering the whole valid range of counters. For example, keeping every kth session key when $k-1$ HMAC computations are done in allowable time period. This helps to balance the time-memory trade-off.

With the assumption that the computation of Keyhandle requires the same keyed hash computation, no special hardware is required for the session key computation and hence no extra storage. We recommend to store the tuple Keyhandle and corresponding *Counter* for each registered relying party at the U2F token. The requirement of this storage is explained in detail in section 7. Considering the existing storage capacity of any hardware token, the overall space overhead is not significant.

2. **Proposed Technique 2:** This technique is proposed by Kocher [34]. It requires two values, an initial secret DS_K (in case of U2F) and the *Counter* i, which gets incremented at each new derivation of the session key K_i. It introduces two reversible functions F_A and F_B and their corresponding inverses as ${F_A}^{-1}$ and ${F_B}^{-1}$ respectively. The function g can be application of functions from among two forward functions (F_A and F_B) and their inverses (F_A^{-1} and F_B^{-1}) in a sequence depending on the value of the *Counter* i. It follows a tree structure. The root is the initial key $K_0 \leftarrow DS_K$ and all the left and right childes are derived using the functions F_A and F_B respectively. To move from the left child to its parent the function ${F_A}^{-1}$ is applied and similarly to move from the right child to its parent the function ${F_B}^{-1}$ is applied. Depending on the position, each session key can be repeated maximum three times. The value D denotes the depth of the tree. Therefore, the number of transactions (considering the repetition of keys) that can be derived before the end of the process is equal to

$$2^{D-1} + \sum_{i=0}^{D-2} 3(2^i) = 2^{D-1} + 3(2^{D-1} - 1) = 2^{D+1} - 3 \qquad \ldots (1)$$

The example case of $D = 5$ which allows $2^6 - 3 = 61$ transactions is shown in Fig 8. The D=5, means that 5 levels of key values are present. Each of the boxes in the figure represents a value of the session key. Thus, multiple dots in a box represent different session keys sharing the same secret value which is maximum three for this technique. For example, if it is required to compute K_6 which is $g(K_5, 6) = {F_A}^{-1}({F_A}^{(5)}(DS_K))$ where $({F_A}^{(5)}(DS_K))$ signifies 5 times recursive use of function F_A starting from initial value DS_K.

Security: This tree based approach for re-keying can be seen as a hybrid approach incorporating the sequential and the parallel approach [16]. The session key is derived by applying a series of computation depending on the value of the *Counter* and the maximum repetition allowed for the session keys is three. Therefore, the leaked information is not sufficient to compromise the secret. The proposal requires the use of a reversible function and any secure block cipher can be used. Following the existing security analysis, it is a secure approach [16].

Usability: The implementation needs four functions $F_A, {F_A}^{-1}, F_B, {F_B}^{-1}$ which can easily be instantiated with a single block cipher like AES computed with the previous session key to generate

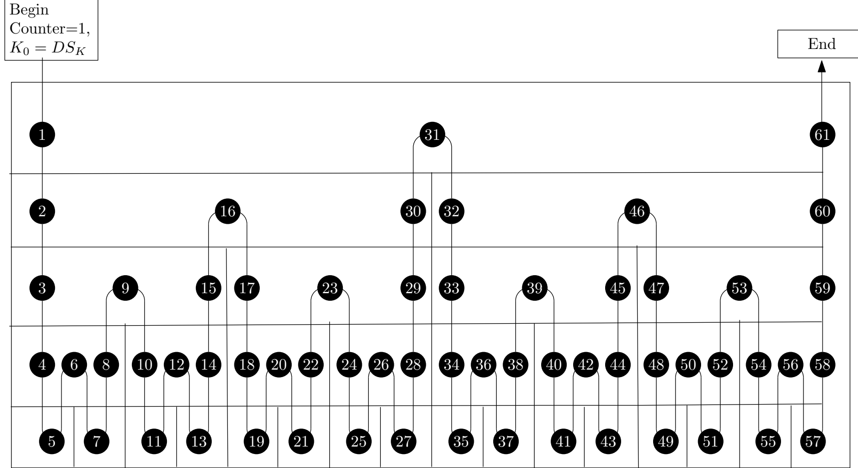


Fig. 8: The diagram shows the function $g(K_{i-1}, i) = K_i$ where $1 \leq i \leq 61$ for $D = 5$ which decides the number of possible session keys as shown in equation 1. The function g can be application of functions from among two forward functions (F_A and F_B) and their inverses (F_A^{-1} and F_B^{-1}) in a sequence depending on the value of the *Counter* i as explained under Kocher's scheme in Section 5. The diagram is taken from [34].

the current session key. At the time of registration, the computation for a session key requires one extra computation when previous value is stored. Therefore computation overhead is negligible. At the time of authentication the maximum number of computation is equal to the depth D of the tree, i.e. the complexity is $log(n)$ where n is the number of nodes of the tree. When $D = 39$, the number of transactions is more than 1 trillion (10^{12}) which means maximum 39 computations to re-derive the session key. This overhead is not significant while with current implementation of AES-128 on Intel Core i7 supporting AES-NI instruction set extensions we can have approximately 2^{29} operations per second [36].

With the assumption that the computation of Keyhandle requires the same block cipher, no special hardware/software is required for the session key computation and hence no extra storage. The storage of all the registered (Keyhandle, *Counter*) pair inside the U2F token is required. Considering the existing storage capacity of hardware token this overhead is not significant.

5.2 Countermeasure to minimize the effect of Attestation Private Key Recovery

The signature function that uses the attestation private key also leaks information that can be captured through side channel analysis as explained in Section 4.2. Usually in public key infrastructure (PKI) [9], the verification of digital signature is satisfied by the use of certificate. Intuitively, it may seem that use of session key will prevent the side channel attack on attestation private key as well. However, getting valid certificate for each session key generated from attestation private key is not practical in asymmetric cryptosystem. Hence, it is difficult to prevent side channel attacks using the existing solutions for signature. Therefore, we suggest a modification in the current approach for assigning the attestation private key to the U2F token. The approach can not prevent the recovery of the attestation private key but minimizes the effect of the attack. Before explaining our proposed recommendation, we explain the existing issues with attestation private key for U2F token.

5.2.1 Trust on FIDO The attestation certificate Acert to prove the genuineness of the U2F token is issued to the manufacturer by some trusted CA. The FIDO server maintains a centralized database of all valid certificates. One of the goals as mentioned in the specifications [12,13] for U2F is to make it a standard solution. A standard should be accepted by all. This means FIDO needs to acquire the global acceptance with a trust on this centralized approach. However, it is difficult to develop trust worldwide. One possible solution could be a decentralized approach. If a government agency under each country takes the responsibility to maintain a server with trusted list of CAs and list of valid certificates, the citizens can easily trust it. The

FIDO server should also include all the valid certificates, then U2F solution could be used as a standard. To prove the genuineness, the relying party can verify with country-level trusted server or the FIDO server or both. This way U2F could be a standard solution accepted world-wide.

5.2.2 Genuineness or Anonymity? In the specifications [12, 13], it is mentioned that a manufacturer of a U2F token provides an attestation certificate signed by some trusted CA along with the attestation private key, burned inside the token. The attestation certificate and signature with signing key proves the genuineness of manufacturer not the token. This is because a large bundle of U2F tokens are provided with the same attestation private key and corresponding certificate to preserve anonymity of the token. However, if a single device is compromised, then all tokens with same identity are compromised. This can be prevented if each device has it's own attestation public-private key pair. This attestation certificate is required to show genuineness of the token. The reason behind the same attestation private key with a large number of tokens is to maintain anonymity. As U2F is second factor and user association is already established with the username and password, there is no enhanced security obtained with anonymity of the token. Therefore, token should only prove its legitimacy which is the main goal towards this approach. Therefore, we see a trade-off between security and anonymity in the case of U2F protocol. As single compromise of attestation private key impacts a large number of tokens, therefore security should be preferred over anonymity.

5.2.3 Countermeasure to minimize the effect of Attestation Private Key Recovery The main idea behind the 'private key' of asymmetric cryptosystem is to keep it private with the owner and should not be shared with any third party. However, the scenario of sharing same attestation private key with a huge number of tokens can be visualized as the manufacturer keeping its own private key to all tokens manufactured by it. One can also visualize this scenario as the identity of the manufacturer associated with the attestation private key which is shared with large number of users of the token, not private to the manufacturer. While private key should be private to its owner. Therefore the manufacturer should not distribute the same attestation private key among a large number of U2F tokens. Instead, it should have its own private key and corresponding certificate signed by a trusted CA. A unique public-private keypair should be generated by the manufacturer for each token. To prove genuineness in this scenario, the manufacturer should sign the public key of the token. The signature by the manufacturer together with the certificate by a trusted CA including the manufacturer public key proves the genuineness of the U2F token. This two layers of signature prevents the sharing of private key and proves the genuineness of the U2F token which is the goal behind the attestation concept of U2F tokens. This also enhances the security as attacker has to forge two signatures now. The unique attestation private key is still vulnerable to the attack which is shown as Attack point-II in Fig. 6 but limited to target a single U2F token at a time instead of a large number of tokens.

5.3 Overview of the U2F protocol with Our Proposed Modifications

In this section, we briefly explain the U2F protocol incorporating the proposed countermeasure and the recommendation.

5.3.1 Our Modified U2F Manufacturing Phase: Unique Secrets for U2F Token This phase is executed at the time of manufacturing of the U2F token as shown in Fig 9. In this phase the U2F token is provided with a randomly generated unique secret key called Device Secret Key DS_K at Step1. At Step2, a unique public-private key pair (PK_T, SK_T) is randomly generated from a key generation function $\mathcal{K}(.)$ for the token. The asymmetric keypair of the manufacturer is represented as (PK_M, SK_M). At Step3, a certificate Acert which includes the key PK_M (public key of the manufacturer) is provided to the U2F token. The Acert is issued by a trusted CA to the manufacturer. At Step4, a signature $\mathcal{S}_M$ including the key PK_T which is signed by the manufacturer signing key SK_M is provided. The certificate Acert includes PK_M and hence helps to verify the signature $\mathcal{S}_M$ of the manufacturer. The signature $\mathcal{S}_M$ by the manufacturer with SK_M verifies the genuineness of the U2F token which is the goal behind the attestation concept for U2F token. Both DS_K and (PK_T, SK_T) are fixed and unique for a token. A global variable *ctr* initialized to zero is provided at Step5. This helps to detect cloning at the authentication phase. A global variable *Counter*

is provided at Step6 which is initialized at the value one and used to generate session keys from DS_K to prevent repeated use of DS_K at the following registration phase.

5.3.2 Our Modified U2F Registration Phase: Use of Session Key The registration process is explained in Fig. 10. At Step1, the username and password are provided to the the server. On verification of the received values the server generates a r-bit random value R at Step2. At Step3, the value R and server AppId are send to browser. Adding the ChannelId, the browser forwards the received values to the U2F token at Step4. At Step5, the U2F token first generates the site-specific key pairs (PK_u, SK_u) applying a random key generation function $\mathcal{K}(.)$. The value of *Counter* is assigned to a variable i which gets incremented after each registration. It then computes the session key K_i following the steps of Algorithm 1. The session key generation function $g(K_{i-1}, i)$ can be instantiated with either the proposed hashed based approach or using the Kocher's technique as explained in Section 5 and shown in Fig. 7 and 8 respectively. Next the Keyhandle Kh_i is generated from the function $f_{K_i}(.)$ following one of the approaches as explained in Fig. 3 and 4. Finally the token adds the values (PK_u, SK_u, Kh_i, i) in a database represented as U2F_db inside it. It computes a signature $\mathcal{S}$ with the key SK_T on the values (c, PK_u, Kh_i, i). At Step6 the token sends the values PK_u, Kh_i, i, Acert, $\mathcal{S}_M$ to the browser. The browser forwards the received values to the server at Step7. On receiving the values the server first verifies the signature $\mathcal{S}$ with PK_T after verification of $\mathcal{S}_M$ with Acert. If verified it keeps the values $(u, H(p \parallel s), PK_u, Kh_i, i$, Acert, $\mathcal{S}_M)$ in its database, else it rejects the registration request.

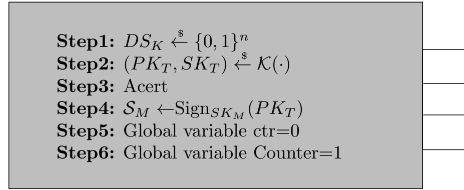


Fig. 9: **Our Modified U2F Manufacturing Phase**, DS_K is a randomly generated n bit device secret. $\mathcal{K}(.)$ is a random key generation function to generate a unique asymmetric keypair (PK_T, SK_T) for the token. The asymmetric keypair of the manufacturer is (PK_M, SK_M). Acert is the certificate issued by a trusted CA which contains PK_M. $\mathcal{S}_M$ is a signature on PK_T by the manufacturer with signing key SK_M. At registration phase, a signature with SK_T is verified with PK_T which is again verified from $\mathcal{S}_M$ and Acert. The global variable ctr is initialized at value zero which is incremented after each successful authentication as explained in Section 2.2.3. The global variable *Counter* is initialized at value one which is incremented after each successful registration as explained in Section 2.2.2.

5.3.3 Our Modified U2F Authentication Phase At the time of authentication, receiving the request with username u and password p at Step2, the server checks for the corresponding registered values. If registered, the server sends the values AppId, Challenge R, Keyhandle Kh_i and *Counter* i to the browser at Step3. The bowser forwards the received values along with TLS ChannelId to the U2F token at Step4. At Step5 the U2F token first checks for the entry of the pair (Kh_i, i) in its database. If the entry is verified, it re-derives the session key K_i from function $g(K_{i-1}, i)$ following the re-keying technique explained in Section 5 and then verifies the received value Kh_i. If the function is ${f_{K_i}}^{-1}(.)$, the U2F token computes ${F_{K_i}}^{-1}(Kh_i)$ which provides the output (AppId$'$ $\parallel SK'_u$). It then verifies if AppId$'$ = AppId. Else for the case when the function is $f_{K_i}(.)$, it computes F_{K_i}(AppId $\parallel SK_u$)$= Kh'_i$ and verifies if $Kh'_i = Kh_i$. Both the cases are shown in Fig. 3 and 4 respectively. On verification for both cases, the token increments a counter, we represent it as 'ctr' which is initialized at value zero as mentioned in [12, 13]. The value ctr is incremented and at Step6, a signature $\mathcal{S}$ including ctr are send to the browser by the U2F token. The browser forwards the received values to the server at Step7. Finally at Step8 the server verifies the signature $\mathcal{S}$ with PK_u and after verification it updates the value of ctr else the authentication is rejected.

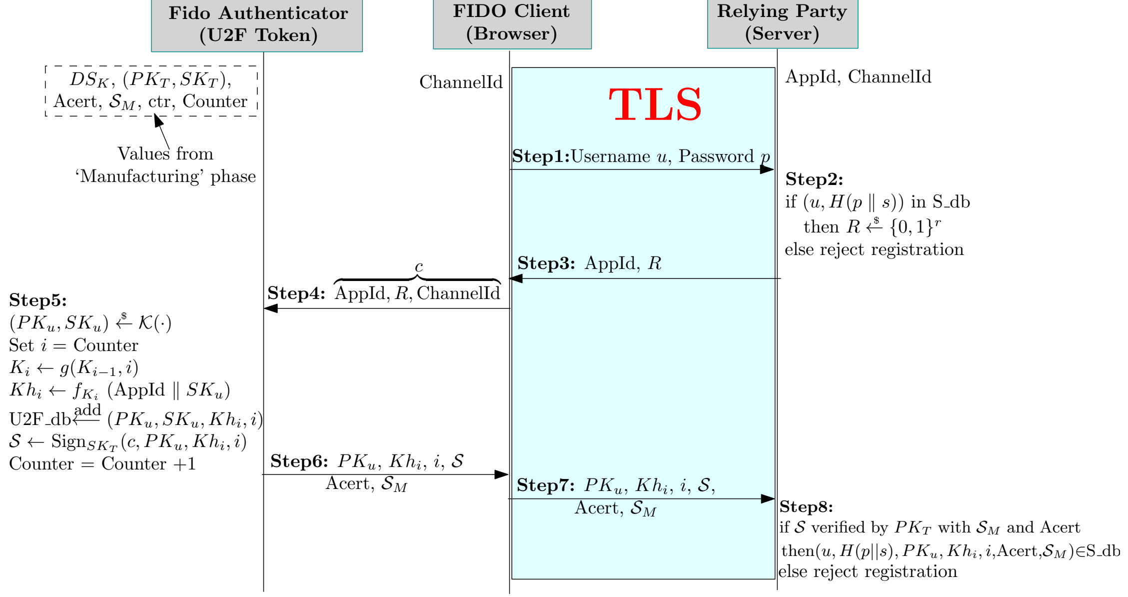


Fig. 10: **Our Modified Registration Phase.** the key DS_K, the asymmetric keypair$(PK_T,\ SK_T)$ with a signature $\mathcal{S}_M$ containing PK_T, Acert, counters 'ctr' and *Counter* are from the manufacturing phase. AppId is the URL of the server. ChannelId is the TLS ChannelId. S_db is the server database described in Section 6.4. R is a r bit random number generated by the server. $\mathcal{K}(.)$ is a key generation function to generate site-specific random keypairs (PK_u, SK_u). The function g generates the session key K_i as shown in Fig. 7 and 8. $f_{K_i}(.)$ is the function to generate Keyhandle Kh_i as shown in Fig. 3 and 4, U2F_db is the database at U2F token as explained in Section 6.3.

6 Design Rationale

In this section we analyse the role of both the counter values, namely *Counter* and 'ctr' and the databases S_db and U2F_db for the execution of our modified U2F protocol.

6.1 Role of *Counter*

The global variable *Counter* is initialized at 1 and gets incremented after each successful U2F token registration to a website. Specifically the value *Counter* is crucial to generate the value Keyhandle as shown in Fig. 10. The value of *Counter* signifies the number of entries at the U2F database U2F_db as explained in Section 6.3. As described in Fig. 10, the counter i helps to generate the session key K_i and from K_i the Keyhandle Kh_i is generated. Hence, the value of *Counter* determines the value of the key K_i. Therefore, the pair (Kh_i, i) corresponding to a username u is maintained in both the server and U2F token databases. At the time of authentication the server sends the pair (Kh_i, i) to the U2F token. Before initiating any computation the U2F token first checks for the valid entry of the (Kh_i, i) pair in its database. It proceeds only if the entries match.
Problem with invalid (Kh_i, i) pair: At the time of authentication, an attacker can provide the *Counter* i which generates a specific session key K_i but incorrect Keyhandle Kh_i. Multiple such attempts by the attacker with different incorrect Keyhandles provide multiple traces operated under same key K_i. This is possible because the value of the session key directly depends on the value of the *Counter*. Therefore, the attacker can collect multiple traces corresponding to different input (Keyhandle)-output pairs computed under same session key. To prevent such malicious attempt, it is required to allow only the legitimate (Kh_i, i) pairs to initiate the corresponding computation. Therefore, we recommend to keep all the registered (Kh_i, i) pairs in a database inside the U2F token. It is secure as compromising the data inside the U2F token is assumed to be difficult to achieve. When the valid pairs are provided to the token, it initiates the corresponding computation, else the authentication is refused. Hence, proposed solution prevents the

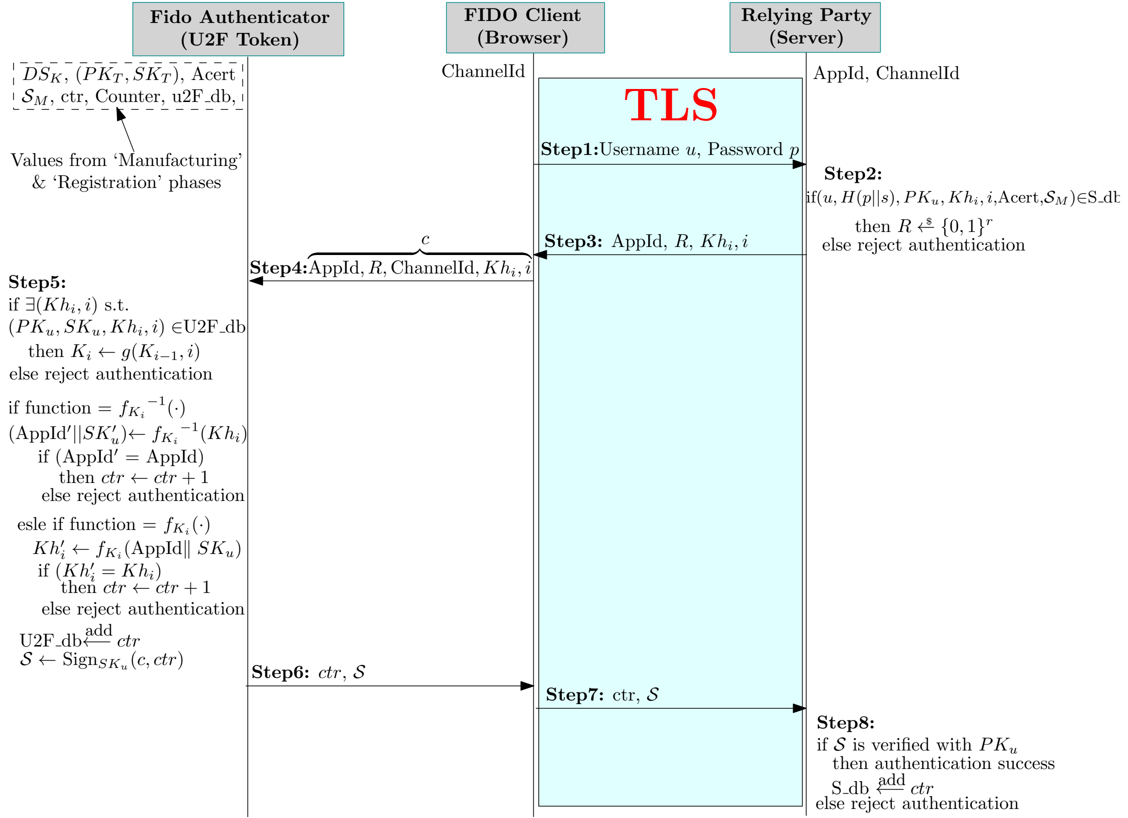


Fig. 11: **Our Modified Authentication Phase.** The key DS_K and the asymmetric keypair(PK_T, SK_T) with signature $\mathcal{S}_M$ which includes PK_T, Acert, the counters 'ctr' and *Counter* are from manufacturing phase. The database U2F_db as explained in Section 6.3 contains all registered values from registration phase. S_db is the server database explained in Section 6.4. R is a r bit random number generated by the server. AppId is the ULR of the server. ChannelId is the TLS ChannelId. Kh_i is the Keyhandle corresponding to the *Counter* i. The function $g(K_{i-1}, i)$ generates the session key K_i as shown in Fig. 7 and 8. The function $f_{K_i}(.)$ is shown in Fig. 3 and 4. 'ctr' is the counter used to record the number of authentication requests successfully satisfied by the U2F token.

collection of different traces corresponding to different input-output pairs computed under same key.

6.2 Role of ‘ctr’

The value ‘ctr’ can be a global or a local (specific to a website) variable as explained in [12, 13]. It gets incremented after each successful authentication phase of the U2F protocol. In this work we consider the ‘ctr’ as a global variable. The value ‘ctr’ helps to detect cloning though it is not a secure approach towards that goal. Specifically, after each successful authentication the value of the ‘ctr’ gets incremented and the server keeps the current value in its database. If the received value of ‘ctr’ at Step8 of Fig. 11 is less than the last recorded ‘ctr’ at the server database, it shows a chance of cloning of the U2F token. This way a simple use of counter ‘ctr’ may help to detect cloning of the U2F token which is in practice not easy to detect or prevent.

6.3 Role of U2F_db

The database at U2F token is represented as U2F_db. It contains the value generated at registration phase corresponding to each registration. For each registration the U2F_db keeps a tuple including site-specific keypairs (PK_u, SK_u), Keyhandle Kh_i corresponding to *Counter* i, for instance the tuple $(PK_u, SK_u,\ Kh_i,$ $i)$ as shown in Fig. 10. The value of *Counter* shows the total number of entries in the U2F_db. The use of database can not be optional to provide a secure authentication against side channel attack. To make the protocol side channel resistant, it is required to allow only the legitimate computations inside the U2F token. Therefore, valid entries are verified from the U2F_db before initiating any computation. At our modified authentication phase information leakage is possible due to illegitimate computation providing wrong counter and Keyhandle pair (Kh_i, i) as explained in Section 6.1. This can easily to prevented by verifying the existence of the pair, for example, (Kh_i, i) in the U2F_db before initiating the computation.

6.4 Role of S_db

The database at the server is represented as S_db. It contains the values generated at registration and authentication phase corresponding to each registered username u. For each username it keeps a tuple including hash of the password with salt $H(p \parallel s)$, public key PK_u for u, Keyhandle Kh_i and corresponding to *Counter* i, the certificate Acert, the signature $\mathcal{S}_M$ and the value of ‘ctr’ to record number of successful authentication corresponding to the U2F token i.e., a tuple $(u, H(p \parallel s), PK_u,\ Kh_i,\ i,$ Acert, $\mathcal{S}_M$, ctr) as described in Fig. 11. At the time of authentication phase, if the server receives the value of ‘ctr’ from the U2F token which is less than the value of ‘ctr’ in S_db, it marks the U2F token as compromised. It shows the higher chance of cloning of the device. The values from S_db when verified through successful execution of our modified authentication phase, user gets access to the server database.

7 Evaluation

We evaluate our proposed modified U2F protocol considering different scenarios under the modified registration and authentication phases. We evaluate considering different combination of values that can be tampered and effect the security of the overall system.

7.1 Modified Registration Phase

At this phase the values forwarded by the browser to the U2F token at Step3 contains AppId, ChannelId and random challenge R. Only the value AppId is useful for the computations of the values, specifically the Keyhandle which is used at the time of authentication phase.

Scenario-I (AppId, R, ChannelId) This is the case of authentic request for registration. Valid Kh_i would be registered when corresponding to *Counter* is i. Multiple registration request with legitimate inputs may reveal the signing key SK_T as discussed in Section 4 through side channel, however single signature is safe.

Scenario-II (AppId′, R, ChannelId) This is the case when authentic AppId is replaced with tampered AppId′ at the time of registration request. Therefore, at U2F token, Keyhandle would contain AppId′ not AppId. Hence, authentication to fack AppId′ without detection is possible when everytime AppId is replaced to AppId′ else it shows DoS attack for legitimate AppId.

7.2 Modified Authentication Phase

We evaluate possible scenarios by tampering the values (AppId, Kh_i, i) corresponding to *Counter*=i. These values are provided by the browser to the U2F token at Step4 of Fig. 11 and further verification of Step5 is performed using them.

Scenario-I (AppId, Kh_i, i): This is the case when valid inputs corresponding to *Counter* i is provided to U2F token. As values are not tampered computations of Step5 satisfies validation and hence are further Steps. This is the scenario of successful authentication at U2F token. Further issue is covered under Scenario-IX.

Scenario-II (AppId, Kh_i, i'): This is the case when *Counter* i corresponding to the Kh_i is tampered to value i'. In this scenario if the check for entries at U2F_db is not performed, incorrect computation corresponding to i' produces $K_{i'}$ and corresponding computation of $Kh_{i'}$. The detection is possible with the verification of Kh_i having legitimate AppId as explained in Fig. 11. When multiple request with i' and different legitimate (AppId, Keyhandle) pair are provided, such unwanted computation leaks information about the key $K_{i'}$. By compromising $K_{i'}$ an attacker can reveal all K_is where $i > i'$. Which also discloses the value of SK_u when function is ${f_{K_i}}^{-1}(.)$ as shown in Fig. 3. After registration phase, the signature at each authentication phase is performed with SK_u. Therefore, it is easy to simulate functionality of the U2F token including SK_u without possession of the token. However, new registration knowing the SK_u can not be verified as at the time of registration the signing key is SK_T. Therefore, verification with database is required to restrict such malicious attempt.

Scenario-III (AppId, Kh'_i, i): This is the case when Kh_i corresponding to *Counter* i is tampered to value $Kh_{i'}$. In this scenario if the check for entries at U2F_db is not performed, computations corresponding to i produces K_i and then Kh_i. The detection is possible with the check that $Kh_i \neq Kh_{i'}$. This is a case of Denial of Service (DoS) attack after initiating unnecessary computations.

Apart from DoS attack, this scenario similar to Scenario-II can compromise the key K_i and further SK_u. This is possible when an attacker obtains multiple traces corresponding computations with key K_i and different Kh'_i. As early detection is always preferred, verifying with database entry is required to allow only valid computations.

Scenario-IV (AppId, Kh'_i, i'): This is the case when both the Keyhandle and the *Counter* values are tampered. Without the check of valid entries both the scenario of II and III are possible. Hence verification with database entry to allow valid computations is required.

Scenario-V (AppId′, Kh_i, i): This is the case when AppId corresponding to valid registered Kh_i, i is tampered to other value AppId′. This fails to satisfy the verification computation of Step5. Hence, The scenario comes under the DoS attack. Hence database entry to stop unnecessary computations is required.

Scenario-VI (AppId′, Kh_i, i'): This is the case when *Counter* i corresponding to the Kh_i is tampered to value i' and AppId to AppId′. In this scenario if the check for entries at U2F_db is not performed, incorrect computation corresponding to i' produces $K_{i'}$ and corresponding computation of $Kh_{i'}$. The detection is possible with the verification of Kh_i and AppId. Such unwanted computation leaks information about the $K_{i'}$. Multiple such attempts with valid Keyhandle but incorrect i' leak information about $K_{i'}$. Compromising $K_{i'}$ an attacker can reveal all K_is where $i > i'$. Which also discloses the value of SK_u as explained under Scenario-II. Hence, software simulation of the U2F authentication phase is feasible. Therefore, verification with database entry is required to allow only valid computations.

Scenario-VII (AppId′, Kh'_i, i): This is the case when Kh_i corresponding to *Counter* i is tampered to value Kh'_i and AppId to AppId′. In this scenario if the check for entries at U2F_db is not performed, computation corresponding to i produces K_i and then Kh_i. The detection is possible with the check that $Kh_i \neq Kh_{i'}$. This is a case of DoS attack after performing unnecessary computations. Another possible attack through side channel is by collecting traces from the computations corresponding multiple such invalid Keyhandle and valid *Counter* i pairs. It compromises the key K_i and hence SK_u as

explained under Scenario-II. Therefore, verification with database entry is required to allow only valid computations.

Scenario-VIII (AppId$'$, Kh'_i, i'): This is the case when all the three values, the Keyhandle, the *Counter* and the AppId are tampered. Without the check of valid entries both the Scenarios of II and III as explained above are possible. The repeated incorrect i' may leak information about $K_{i'}$ and all K_is where $i > i'$. When correct Kh_i corresponding to i is known, SK_u can be disclosed when function is ${f_{K_i}}^{-1}(.)$ as shown in Fig. 3. This helps to simulate the future authentication by the token. Hence database entry to allow valid computation is required.

Scenario-IX (AppId, Kh_i, i, R, ChannelId) The value of R is fixed for a particular session, hence it changes at each authentication request. When the values AppId, Kh_i, i are genuine then all computations of Step5 of Fig. 11 are verified and finally the signature $\mathcal{S}$ is computed including the value of R. As R is one-time, an attacker with access to the U2F token can initiate multiple computations corresponding to the legitimate values but every time gets signature on different input (as R varies). Therefore, signature with SK_u on different input provides different output. Collection of power traces corresponding to different input-output pairs again leaks information about the key SK_u. This helps an easy simulation of the authentication without the possession of the token. The only possible solution is to restrict the use of token by an attacker.

8 Limitations of U2F

At the authentication phase of the U2F protocol, a signature $\mathcal{S}$ is obtained with site-specific signing key SK_u as shown at Step5 of Fig. 11. The values to sign are fixed except R which is randomly generated at each authentication request initiated by the token. As explained above under Senario-IX, this changing R provides different input-output pairs computed under same key SK_u which may leak the value of SK_u through side channel attack. Therefore it is easy for an attacker to initiate a legitimate authentication request multiple times and compromise the key. Then it can simulate the legitimate authentication without possession of the token. This attack is not possible to prevent with existing U2F solution. The possible countermeasure is to restrict the computation only to genuine users of the token. One possibility is to take biometric input to first authenticate a legitimate user and then initiate the further computations. Therefore, the scenario above shows a limitation of U2F solution. Another limitation is transaction non-repudiation as mentioned in the specifications [12, 13] and explained in Section 3.

9 Possible Side Channel Attacks on Universal Authentication Framework (UAF) Protocol and Mitigation

The UAF protocol is an authentication protocol proposed by FIDO alliance which supports biometric authentication to provide a unified and extensible authentication mechanism that supplants passwords [8]. It involves five entities as shown in Fig. 12. The user of the protocol, a UAF authenticator which is similar to U2F token but also includes a database and the verifier to authenticate users biometric input. The ASM which is the software to work as an interface between the UAF authenticator and the browser. The user browser which is called the FIDO client, the relying party which is the web server and the FIDO server which runs on the relying party's infrastructure and maintains a database extracting the data coming from the UAF authenticator. Similar to the U2F protocol, the UAF protocol can be explained in three phases: 1. *The UAF Manufacturing Phase* that mainly assigns device secret key DS_K and an asymmetric keypair (PK_M, SK_M) to the UAF authenticator, 2. *The UAF Registration Phase* which registers a user for future authentication and 3. *The UAF Authentication Phase* which authenticates the registered users. Following is the description of the three phases of the UAF protocol.

9.1 UAF Manufacturing Phase

This phase is executed at the time of manufacturing of the UAF token as shown in Fig 13. In this phase the UAF token is provided with a randomly generated unique secret called Device Secret Key DS_K at Step1. At Step2, a public-private key pair (PK_M, SK_M) is provided to the token. The (PK_M, SK_M) keypair is not a

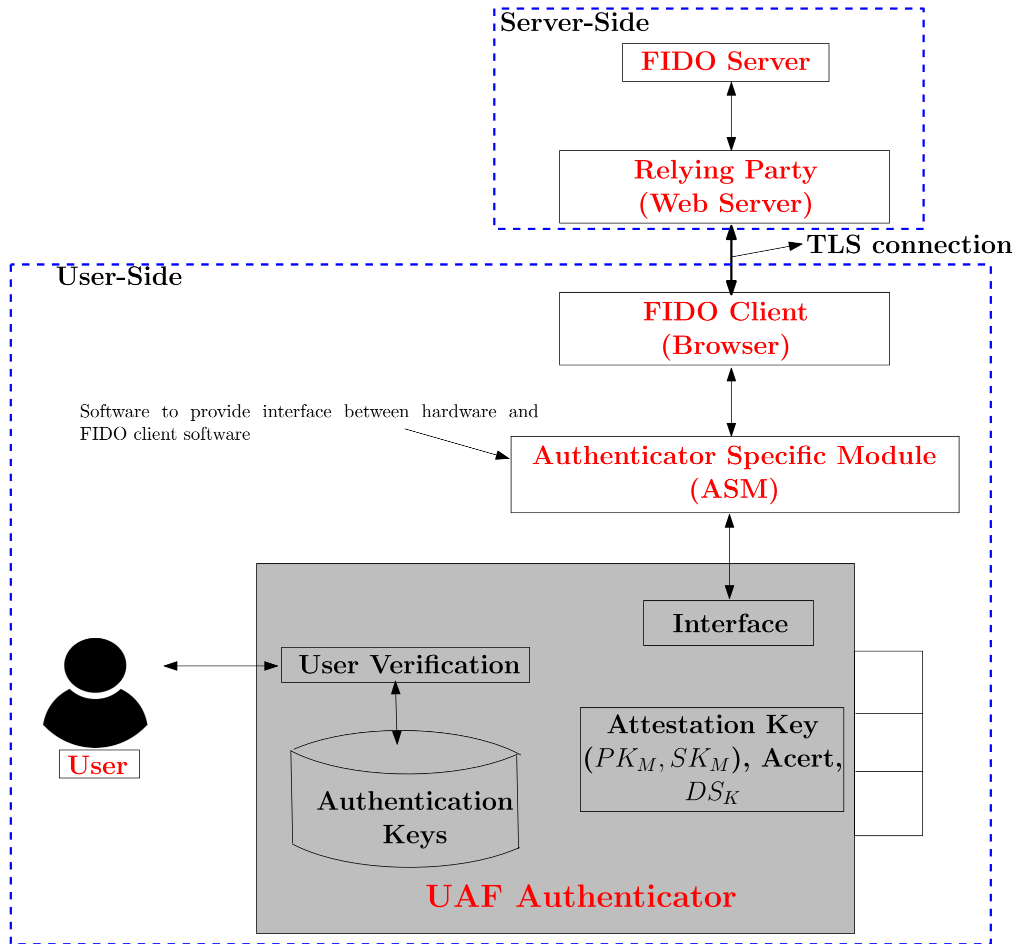


Fig. 12: **UAF protocol entities.** UAF authenticator is the UAF token provided by manufacturer. The key DS_K, the asymmetric keypair(PK_M, SK_M) with attestation certificate Acert which includes PK_M are provided by the manufacturer. ASM is the software to provide interface between hardware token and client browser.

one-time generation. Once randomly generated from a key generation function, the manufacturer provides the same (PK_M, SK_M) pair to multiple tokens. The scenario can be interpreted as the keypair (PK_M, SK_M) generated for the manufacturer and shared with all the tokens manufactured by it. At Step3, a certificate Acert issued by a trusted CA which includes PK_M (the public key of the manufacturer) is provided to the token. During the registration phase, the signature with SK_M is verified with PK_M which is extracted from Acert by the server. This verification proves the genuineness of the UAF token. Therefore, both DS_K and (PK_M, SK_M) are fixed for a token while the same (PK_M, SK_M) values are provided to multiple tokens which preserve the anonymity of the token. An integer counter 'ctr' is provided at Step4. This 'ctr' is initialized to value zero and gets incremented after each successful execution of the authentication phase.

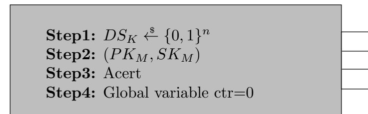


Fig. 13: **UAF Manufacturing Phase.** The key DS_K is a randomly generated n-bit secret. The keypair (PK_M, SK_M) is the asymmetric keypair of the manufacturer shared with the token. Acert is the certificate signed by a trusted CA which includes PK_M. At the time of registration phase, a signature with SK_M is verified with PK_M from Acert by the server to prove the genuineness of the UAF token. The global variable ctr is initialized to zero and incremented after each successful authentication.

9.2 UAF Registration Phase

The registration procedure is explained in Fig 14. Initially the username u and password p are communicated to the FIDO client at Step1 which is forwarded to the server at Step2. At Step3 the server verifies the values u and $H(p \parallel s)$ which is the hash of the password where s is the salt. On successful verification, the server triggers the UAF registration request to FIDO server at Step4. At Step5 the FIDO server generates a r bit random challenge R to differentiate each request. The FIDO server sends the UAF registration policy which includes the authentication modes supported by it and the value R to the server at Step6 which is forwarded at Step7 to the FIDO client. At Step8 the FIDO client forwards the identity of the server as AppId and the value R alongwith the registration policy to the ASM. ASM generates an n-bit random value called the *kid* and computes a value called keyhandle access token (KT) which is the hash of the AppId and *kid*. The ASM stores the pair (*kid*, KT) in its database represented as A_db. At Step9, the ASM forwards the received values of Step8 and the pair (*kid*, KT) to the UAF token. At Step10 the UAF token requests user verification satisfying the UAF registration policy communicated by the FIDO server. On successful user verification, at Step11 the UAF token first generates a website specific public-private key pair represented as (PK_u, SK_u) by applying a random key generation function $\mathcal{K}(\cdot)$. The function $\mathcal{K}(\cdot)$ can be an openSSL key pair generation library which needs as input an elliptic curve such as the NIST standard P-256 curve. We define a key dependent function $f_K(\cdot)$ where K is the secret key. The UAF token computes the function with key DS_K and outputs f_{DS_K} (SK_u, KT, u) which is called the Keyhandle Kh. The above function can be instantiated with any block cipher such as AES in CBC mode or HMAC. It then computes a signature $\mathcal{S} \leftarrow \text{Sign}_{SK_M}(R, \text{AppId, ChannelId, AAID}, kid, PK_u)$. At Step12, the token sends the values PK_u, C, AAId, *kid* and $\mathcal{S}$ along with ACert to the ASM. The ASM forwards the received values to the browser at Step13. The browser forwards the received values to the server at Step14. After receiving the values, the server forwards the values to the FIDO server at Step15. At Step16, the FIDO server first verifies the signature $\mathcal{S}$ with PK_M from the certificate Acert. This verification proves the genuineness of the UAF token. On verification, the FIDO server updates its database FIDO_db with the values (AAId, *kid*) corresponding to the username u and it shows the successful registration of the UAF token else the server rejects the registration request. The UAF protocol allows a user to register multiple tokens with the same account.

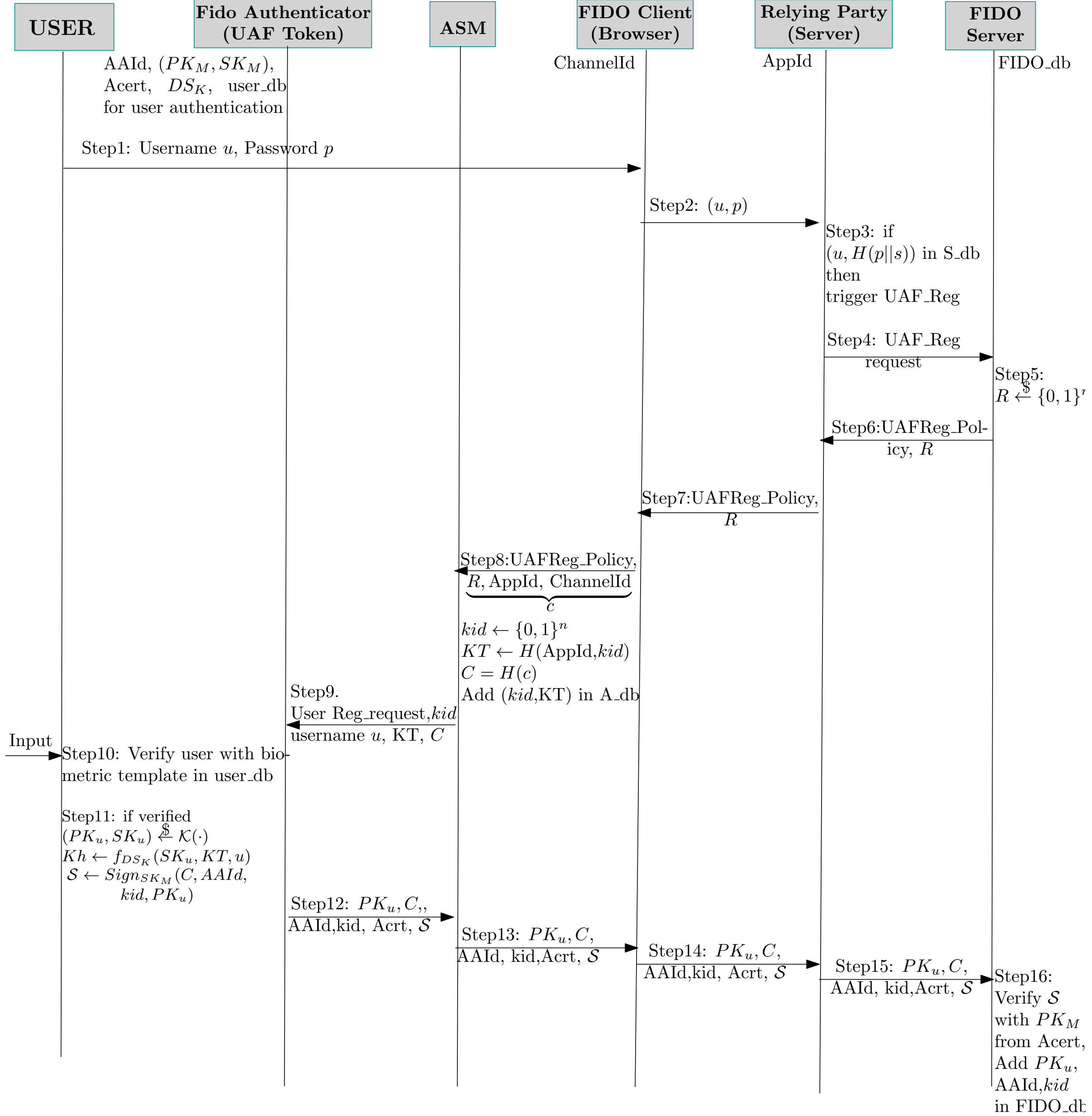


Fig. 14: **UAF Registration Phase.** The key DS_K, the asymmetric keypair(PK_M, SK_M) with attestation certificate Acert which includes PK_M and the counter 'ctr' are from the manufacturing phase. AppId is the URL of the server. ChannelId is the TLS ChannelId. FIDO_db is the FIDO server database. R is a r bit random number generated by FIDO server. $\mathcal{K}(\cdot)$ is a random key generation function which generates the keypair (PK_u, SK_u). $f_{DS_K}(\cdot)$ with key DS_K is the function to generate Keyhandle Kh, A_db is the database of ASM.

9.3 UAF Authentication Phase

The authentication procedure is explained in Fig 15. Once second factor authentication with UAF is registered, for example, when the values (AAId, PK_u, kid, Acert) corresponding the username u are registered with the FIDO server, the subsequent login to the website needs to verify the registered values by communicating with the UAF token through browser. The steps are as follows. After receiving username u at Step1, the browser forwards the value to the server at Step2. On receiving the value, the server requests UAF authentication to the FIDO server at Step3. The FIDO server checks its database to retrieve the values AAId and *kid*. At Step4 the FIDO server generates a r bit random challenge R and sends it along with AAId, *kid* and the policy to the server at Step5. The server appends its AppId and forwards the received values to the browser at Step6. The browser forwards the received information with ChannelId to the ASM at Step7. Checking the policy, the ASM selects the authenticator supported by the attached UAF token. It then computes the keyhandle access token KT from the received AppId and *kid* and a value C which is the hash of values (AppId, R, ChannelId) at Step8. At Step9, the ASM sends the values AAId, KT and C to the UAF token. On receiving the values, the UAF token requests the user verification at Step10. It first verifies the user with stored biometric template at Step11. On successful verification, it verifies Kh corresponding the received KT at Step12. Specificlly, it performs inverse computation ${f_{DS_K}}^{-1}(Kh) = (SK'_u, KT', u')$. It then compares the received KT with the computed KT'. If a successful match happens, the token increments a counter represented as 'ctr'. This 'ctr' can be a global or a local variable. If it is local variable then each AppId gets its own 'ctr' else a single 'ctr' is used across all registered AppId. Throughout the explanation we consider 'ctr' as a global integer variable. The value of the 'ctr' is incremented after each successful authentication by the UAF token. This value 'ctr' is introduced to detect cloning of the UAF token as explained in Section 6.2 for U2F. The UAF token signs the values (AAId, C, ctr) with SK_u which is represented as $\mathcal{S}$ and sends the values $\mathcal{S}$ and ctr to the ASM at Step13. The ASM then forwards the values to the browser at Step14 and similarly the browser forwards to the server at Step15. At Step16 the server sends the values to the FIDO server. Finally the FIDO server verifies the received signature $\mathcal{S}$ with stored key PK_u for the username u from FIDO_db. On successful verification, it keeps the 'ctr' value with the database and this shows the successful completion of the authentication process.

9.4 Attack on UAF

The UAF authenticator, similar to the U2F authenticator, is assigned a unique device secret key DS_K and an asymmetric keypair (PK_M, SK_M) with attestation certificate Acert at the Manufacturing phase by the manufacturer. The key DS_K is used to compute Keyhandle as explained for the case of UAF and can be compromised by applying side channel attack following the explanation of the Section 4.1. Similarly, the attestation private key SK_M which is used as the signing key to perform signature during the UAF registration phase, can be compromised as explained in Section 4.2. We recommend the same countermeasure explained in Section 5 to protect the DS_K in case of UAF and to provide unique (PK_M, SK_M) pair to each UAF authenticator to mitigate the attack on SK_M. Providing side channel attack resistant solution is crucial for any hardware that can be an easy access to any attacker. Therefore, prevention or mitigation of SCA is of utmost importance considering UAF as well.

10 Conclusions

In this work, we observe that the side channel attack is possible on U2F protocol which may compromise the device secret key DS_k and attestation private key SK_M following the explanation of the specifications [12,13]. Both of these keys are assigned to the U2F token at the manufacturing phase of the U2F protocol. This side channel attack can completely break the second factor U2F solution. We suggest countermeasure to protect the DS_k and suggest modification in the protocol to mitigate the attack of compromising the SK_M. We also present the detailed analysis of the U2F protocol including its security analysis. The side channel attacks can be applied to any such secure device that uses a single fixed key either for encryption or signature and hence the UAF protocol is also an easy target of this side channel attack. We provide a brief overview of the UAF protocol showing how the similar solution for U2F can be applied to UAF as well.

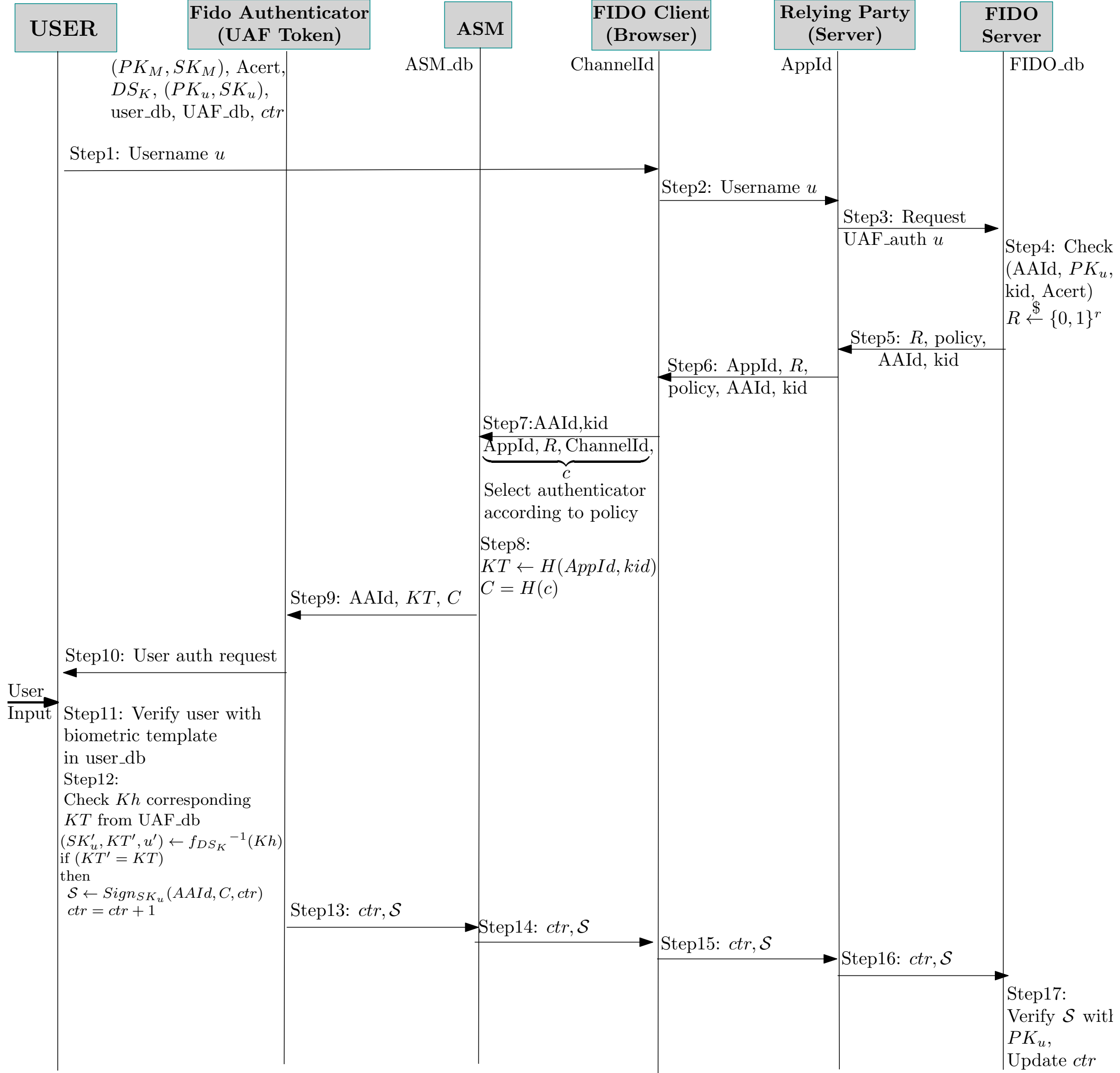


Fig. 15: **UAF Authentication Phase.** The key DS_K, the asymmetric keypair(PK_M, SK_M) with attestation certificate Acert which includes PK_M and counter 'ctr' are provided at the manufacturing phase. The FIDO server database FIDO_db contains all the required values generated at the registration phase corresponding to a registered username. R is a r bit random number generated by the server. AppId is the URL of the server. ChannelId is the TLS ChannelId. The value Kh is the Keyhandle. 'ctr' is the counter used to record the number of authentication requests successfully satisfied by the UAF token.

The open problem for the security of U2F protocol is to find a signature scheme secure against side channel attack. This is an interesting problem which can help to secure many implementations that leak private key through side channel. Working on the countermeasure to overcome the limitations of U2F as explained in Section 8, is another interesting problem to explore.

References

1. Jerome H. Saltzer. Protection and the control of information sharing in MULTICS. In Herbert Schorr, Alan J. Perlis, Peter Weiner, and W. Donald Frazer, editors, *Proceedings of the Fourth Symposium on Operating System Principles, SOSP 1973, Thomas J. Watson, Research Center, Yorktown Heights, New York, USA, October 15-17, 1973*. ACM, 1973.
2. Robert Morris and Ken Thompson. Password Security: A Case History, 1979. `http://cs-www.cs.yale.edu/homes/arvind/cs422/doc/unix-sec.pdf`.
3. Password Hashing Competition (PHC), 2014. `https://password-hashing.net/index.html`.
4. Donghoon Chang, Arpan Jati, Sweta Mishra, and Somitra Kumar Sanadhya. Cryptographic module based approach for password hashing schemes. In Stig Fr. Mjølsnes, editor, *Technology and Practice of Passwords - International Conference on Passwords, PASSWORDS'14, Trondheim, Norway, December 8-10, 2014, Revised Selected Papers*, volume 9393 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2014.
5. TOTP: Time-Based One-Time Password Algorithm. RFC: 6238 , 2011. [Online; accessed 24-May-2017].
6. DRAFT NIST Special Publication 800-63B Digital Identity Guidelines, 2017. [Online; accessed 24-May-2017].
7. Florian Maury and Mickael Bergem. A first glance at the U2F protocol. SSTIC2016, 2016. Available at: `https://www.sstic.org/media/SSTIC2016/SSTIC-actes/a_first_glance_at_the_u2f_protocol/SSTIC2016-Article-a_first_glance_at_the_u2f_protocol-maury_bergem.pdf`.
8. FIDO UAF Protocol Specification. FIDO Alliance Implementation Draft, 02 February 2017. Available at: `https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-protocol-v1.1-id-20170202.pdf`.
9. William Stallings. *Cryptography and network security - principles and practice (3. ed.)*. Prentice Hall, 2003.
10. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
11. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
12. Universal 2nd Factor (U2F) Overview . FIDO Alliance Proposed Standard, 14 May 2015. Available at: `https://fidoalliance.org/specs/fido-u2f-overview-ps-20150514.pdf`.
13. Universal 2nd Factor (U2F) Overview . FIDO Alliance Proposed Standard, 15 September 2016. Available at: `https://fidoalliance.org/specs/fido-u2f-v1.1-id-20160915/fido-u2f-overview-v1.1-id-20160915.pdf`.
14. Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.
15. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
16. Michel Abdalla and Mihir Bellare. Increasing the lifetime of a key: A comparative analysis of the security of re-keying techniques. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 546–559. Springer, 2000.
17. Michael Dietz, Alexei Czeskis, Dirk Balfanz, and Dan S. Wallach. Origin-bound certificates: A fresh approach to strong client authentication for the web. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 317–331. USENIX Association, 2012.
18. D. BALFANZ and R. HAMILTON. Transport Layer Security (TLS) Channel IDs, v01 (IETF Internet-Draf) , 01, 2013. Available at: `http://tools.ietf.org/html/draft-balfanz-tlschannelid`.
19. Wikipedia. Denial-of-service attack, wikipedia, the free encyclopedia, 2017. [Online; accessed 21-April-2017].

20. Nikos Leoutsarakos. What's wrong with FIDO?, 2015. Available at: http://www.zeropasswords.com/pdfs/WHATisWRONG_FIDO.pdf.
21. Wei Wei, Jinjiu Li, Longbing Cao, Yuming Ou, and Jiahang Chen. Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web*, 16(4), 2013.
22. Symantec Internet Security Threat Report trends for 2010. Technical report, Symantec, April 2011. Available at: http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xv_04-2010.en-us.pdf.
23. Internet Security Threat Report 2013. Technical report, Symantec, April 2013. Available at: http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v18_2012_21291018.en-us.pdf.
24. Wikipedia. Non-repudiation — wikipedia, the free encyclopedia, 2016. [Online; accessed 7-June-2017].
25. Jerry Felix and Chris Hauck. System Security: A Hacker's Perspective, September 1987.
26. Wikipedia. Phishing, wikipedia, the free encyclopedia, 2017. [Online; accessed 24-April-2017].
27. Computer Security Division (Information Technology Lab). Guide to Malware Incident Prevention and Handling. NIST Special Publication 800-83 Revision 1.
28. Mauro Conti, Nicola Dragoni, and Viktor Lesyk. A survey of man in the middle attacks. *IEEE Communications Surveys and Tutorials*, 18(3), 2016.
29. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 4-6, 1992*, pages 72–84, 1992.
30. Nikolaos Karapanos and Srdjan Capkun. On the effective prevention of TLS man-in-the-middle attacks in web applications. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 671–686, 2014.
31. Kamin Whitehouse, Chris Karlof, and David E. Culler. A practical evaluation of radio signal strength for ranging-based localization. *Mobile Computing and Communications Review*, 11, 2007.
32. Wikipedia. Side-channel attack, wikipedia, the free encyclopedia, 2017. [Online; accessed 21-April-2017].
33. Tali Garsiel and Paul Irish. How browsers work: Behind the scenes of modern web browsers, 2011. [Online; accessed 4-June-2017].
34. P. Kocher. Leak Resistant Cryptographic Indexed Key Update. US Patent 6539092.
35. Wikipedia. Pseudorandom number generator, wikipedia, the free encyclopedia, 2017. [Online; accessed 4-June-2017].
36. Wikipedia. Advanced encryption standard — wikipedia, the free encyclopedia, 2017. [Online; accessed 8-June-2017].