A Forward-secure Efficient Two-factor Authentication Protocol

Abstract. Two-factor authentication (2FA) is commonly used for controlling access to high-value online accounts, particularly where financial transactions are involved. In such schemes, knowledge factors (*e.g.*, PIN) and hardware device possession are required to complete authentication. However, existing 2FA protocols leave the PIN vulnerable should the server, user's computer, or hardware device be compromised. In this paper, we propose a 2FA protocol proved to be secure against adversaries who can (a) observe the traffic between a user and server and (b) have physical access to the user's device, or its PIN, or breach the server. Compared to previous work, our protocol reduces the cost of authentication devices by not requiring tamper-resistant hardware or that the authentication device be connected to the computer, and through only using efficient symmetric cryptographic primitives. Furthermore, our protocol is highly usable, requiring the user only to remember a short PIN and type short authentication codes; it imposes up to 40% lower communication overhead compared to the state-of-the-art. The protocol achieves these goals through a novel combination of splitting secrets between the server and authentication device and key-evolving symmetric-key encryption.

1 Introduction

Two-factor authentication (2FA) is increasingly required for access to online services, as a way to mitigate the risk of a single factor (typically a password) being compromised. This trend has been accelerated by regulations, such as the Payment Services Directive 2 (PSD 2) [1] in the EU and federal requirements in the US [6]. 2FA can bring significant security benefits, but only if the solution is secure, usable, and cost-effective. However existing systems fall short in these requirements. In this paper, we propose a new 2FA protocol that significantly improves existing systems and so facilitates the wider adoption of 2FA for authenticating access to online services, in particular authorising financial transactions.

2FA requires the combination of two factors out of knowledge (*e.g.*, PIN or a password), possession (*e.g.*, of a hardware device), and biometrics (*e.g.*, a fingerprint). In this paper, we will focus on combining knowledge and possession, since biometrics require that devices have a special sensor and so imposes a significant cost penalty. Biometrics also have shortcomings in that they cannot be effectively revoked, will fail to work for certain people, and create privacy concerns. Our requirements for the 2FA protocol are therefore that it be:

secure: provably capable of preventing unauthorised access in as a wide range of circumstances as possible, including compromise of the server, compro-

mise of the authentication device, compromise of the user's computer, and compromise of the communication network;

usable: create a minimal imposition on the user and environment through not requiring special software to be installed on the user's computer and not requiring that the user remember long passwords or enter long strings into the computer or hardware device; and

cost-effective: the hardware device must be cheap and so must not be required to have tamper-resistant trusted hardware or the ability to perform complex computation

Furthermore, the 2FA protocol must support **transaction authentication** *i.e.*, be able to bind its execution to a particular transaction that is shown to the user on a trusted display, as required by the PSD2. This means that for the authentication protocol to succeed the user must have had the ability to check the transaction details and so be able to detect if malware on their computer has tampered with the transaction details shown on screen.

Researchers and companies have proposed various 2FA solutions based on a combination of PIN and device possession. Some of these solutions offer a strong security guarantee against an adversary which may (a) observe the communication between a user and server, and (b) have physical access to the user's device, or its PIN, or breaches the server. These solutions do not rely on trusted hardware and still ensure that even such a strong adversary cannot succeed during the authentication. Nevertheless, these solutions (i) require a user to remember multiple secret values (instead of a single PIN) to prove its identity which ultimately harms these solutions' usability, (ii) involve several modular exponentiations that make the device battery power run out quickly, and (iii) are proved in the non-standard random oracle model.

Our Contributions. In this work, we present a 2FA protocol that resists the strong adversary described above while addressing the aforementioned shortcomings and imposing a lower communication cost. Specifically, our protocol:

- requires a user to remember and type into the device only a single PIN.
- allows the device to generate a short authentication message.
- does not involve any modular exponentiations.
- is secure in a standard model.
- imposes up to 40% lower communication costs than the state-of-the-art does.

To attain its goals, our protocol does not use any trusted hardware; instead, it relies on a novel combination of the following two approaches. Firstly, neither the server nor the authentication device has the ability to verify the PIN – secret information stored by both is needed to do so. This approach ensures that an adversary cannot retrieve the PIN, even if it penetrates either location. Secondly, it (a) requires that the server and device use key-evolving symmetric-key encryption (*i.e.*, a combination of forward-secure pseudorandom bit generator and authenticated encryption) to encrypt sensitive messages they exchange, and (b) requires that used keys be discarded immediately after their use. This approach

ensures the secrecy of the communication between the parties and guarantees that the adversary cannot learn the PIN, even if it eavesdrops on the parties' communication and subsequently breaks into the device or server. We formally prove the security of this protocol.

2 Related Work

One approach to hardware authentication devices is to plug them into the computer's USB port, but for such devices to work there must be corresponding software installed on the computer, and this might not be possible on shared devices or computers implementing a corporate IT policy. Also, if any special software is needed it would have to be implemented for every supported operating system and processor architecture, and periodically updated, so imposing higher development costs. In the longer term, support for hardware authentication devices may become a common operating system provision, but current proposed standards to do so, such as FIDO 2.0 do not support transaction authentication and are not designed to mitigate compromise of the user's computer. For this reason, connected authentication hardware does not meet the requirements we have set out. Similarly using a smartphone as the hardware authentication device does not fulfil the usability requirement because not everyone has a smartphone or is willing to install special software on it, and smartphones have a large attack surface that can be exploited to extract the authentication secrets.

Instead, we take to follow the approach of a one-time-password (OTP) generator, where the hardware device yields a short string (*e.g.*, six to eight digits) from a combination of a PIN and the device's parameters. Typically, as in the "Open AuTHentication" (OATH) family of protocols the OTP is computed as the keyed hash of these parameters. The user types this OTP into the computer and it is verified by the server, which knows the corresponding key. There are a wide variety of approaches to OTP generation, such as whether a counter (*e.g.*, OATH-HOTP [12]), timestamp (*e.g.*, OATH-TOTP [14]), or server-generated challenge (*e.g.*, OATH-OCRA [13]) is used to limit the validity of a given OTP.

There is also a range of options on how the PIN is used as the knowledge factor to complement the possession factor of the hardware device. One option is to store the PIN on the device and use trusted tamper-resistant hardware to verify that the PIN entered is correct before generating an OTP, while not allowing the correct PIN to be extracted. However, tamper-resistant hardware adds significantly to costs and can be bypassed by a sufficiently well-equipped adversary. Alternatively, as with OATH-OCRA, the PIN can be an input to the keyed hash and verified by the server, but this would allow the PIN to be obtained if the server is compromised. Additionally, if the adversary captures one OTP and extracts the key from the hardware device they would be able to brute-force a low-entropy PIN.

Our protocol is similar to OATH-OCRA in that it is a challenge-response authentication protocol, but unlike OATH-OCRA we protect the PIN from compromise even if the adversary can obtain secrets stored on the server or device. This threat model is stronger than that used for most research and implementation but Jarecki *et al.* [7] do address this threat model. However, this protocol

imposes high costs due to the use of public-key cryptography and numerous rounds of communication. This protocol requires the user (in addition to remembering its PIN) to locally store a cryptographic secret key. The protocol we propose does not require trusted tamper-resistant hardware, needs a single round of communication, and involves no public-key cryptography. A more detailed comparison of the protocols and a survey of the related work can be found in Section 7 and the paper's full version [2] respectively.

3 Notations and Preliminaries

3.1 Notations

To disambiguate the different uses of keys and other items of data, variables are annotated with a superscript to indicate their origin. $\cdot^U$ indicates data stored at the user, $\cdot^S$ means data stored at the server, and $\cdot^M$ indicates data item has been extracted from a message. Table 1 presents a summary of variables.

Table 1: Notation used in the protocol.

Symbol	Purpose	Source and lifetime
$\mathsf{PRF}_k(.)$	Pseudorandom function taking a key k.	Used to derive a verifier and session key.
FS-PRG	Forward-secure Pseudorandom Bit Generator.	Used to derive temporary keys.
k^U, k^S	Authenticated Encryption (AE) key at the user and server sides respectively.	Key k randomly generated by the system operator and stored by the user as k^U and server as k^S at device creation. Constant for the lifetime of the device.
st^U, st^S	The state of FS-PRG at the user and server sides respectively.	Initialised to randomly generated state st_0 at device creation. Updated using FS-PRG.
kt_1^U, kt_1^S	Temporary key for the enrolment phase.	Output by FS-PRG and used for a single message exchange before being discarded.
kt_2^U, kt_2^S, kt_3^U, kt_3^S	Temporary keys of PRF, used in the authentication phase.	Output by FS-PRG and used for a single message exchange before being discarded.
ct^U, ct^S	Counter for synchronising FS-PRG state and detecting replayed messages.	Initialised to zero at device creation. ct^U and ct^S are updated atomically along with st^U and st^S respectively.
N^S, N^M	Random challenge for detecting replayed messages.	Generated randomly by the server for each message.
sa^U	Random PIN-obfuscation secret key.	Initialised to randomly generated value at device creation. Not known by the server or system operator.
PIN^U	User's PIN.	Entered by the user. It is never stored in the device and used to generate a verifier.
v^U, v^S, v^M	Verifier, generated from PIN-obfuscation key and the user's PIN.	Stored by the server after the enrolment phase. It is not stored by the user.
t^S, t^M	Description of a transaction to be authenticated.	Generated and sent by the server.
$response^U$	Authentication response.	Computed by the user.
$expected^S$	Expected authentication response.	Computed by the server.

3.2 Keyed Pseudorandom Function

Informally, a keyed pseudorandom function (PRF) is a deterministic function that takes as input a key and some argument. It outputs a value indistinguishable

from that of a truly random function with the same domain and range. A formal definition of a PRF is given by Katz and Lindell [8].

3.3 Forward-Secure Pseudorandom Bit Generator

A Forward-Secure Pseudorandom Bit Generator (FS-RPG), is a *stateful* object which consists a pair of algorithms and a pair of positive integers, *i.e.*, $\mathsf{FS\text{-}RPG} = \Big((\mathsf{FS\text{-}RPG.KGen},\ \mathsf{FS\text{-}RPG.next}), (b, n)\Big)$, as defined in [4]. The probabilistic key generation algorithm FS-RPG.KGen takes a security parameter as input and outputs an initial state st_0 of length s bits. FS-RPG.next is a key-updating algorithm which, given the current state st_{i-1}, outputs a pair of a b-bit block out_i and the next state st_i. We can produce a sequence $out_1, ..., out_n$ of b-bit output blocks, by first generating a key $st_0 \xleftarrow{\$} \mathsf{FS\text{-}RPG.KGen}(1^\lambda)$ and then running $(out_i, st_i) \leftarrow \mathsf{FS\text{-}RPG.next}(st_{i-1})$ for all $i, 1 \leq i \leq n$. As with a standard pseudorandom bit generator, the output blocks of this generator should be computationally indistinguishable from a random bit string of the same length. The additional property required from a FS-RPG is that even when the adversary learns the state, output blocks generated before the point of compromise remain computationally indistinguishable from random bits. This requirement implies that it is computationally infeasible to recover a previous state from the current state.

Recall, FS-RPG.next updates the state of the forward-secure generation by one step; however, our protocol sometimes needs to invoke FS-RPG.next multiple times sequentially. Thus, for the sake of simplicity, we define a wrapper algorithm $\mathsf{Update}(st_a, d)$ which wraps FS-RPG.next. Algorithm Update as input takes a current state (similar to FS-RPG.next) and new parameter d that determines how many times FS-RPG.next must be invoked internally. It invokes FS-RPG.next d times and outputs the pair (out_b, st_b) which are the output of FS-RPG.next when it is invoked for d-th time, where $b > a$.

3.4 Authenticated Encryption (AE)

Informally, authenticated encryption $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an encryption scheme that simultaneously ensures the secrecy and integrity of a message. It can be built via symmetric or asymmetric key encryptions. In this work, we use authenticated symmetric-key encryption, due to its efficiency. Gen is a probabilistic key generating algorithm that takes a security parameter and returns an encryption key k. Enc is a deterministic encryption algorithm that takes the secret key k and a message m, it returns a ciphertext M along with the corresponding tag t. Dec is a deterministic algorithm that takes the ciphertext M, the tag t, and the secret key k. It first checks the tag's validity, if it accepts the tag, then it decrypts the message and returns $(m, 1)$. Otherwise, it returns $(., 0)$.

The security of such encryption consists of the notion of secrecy and integrity. The secrecy notion requires that the encryption be secure against chosen-ciphertext attacks, *i.e.*, CCA-secure. The notion of integrity considers existential unforgeability under an adaptive chosen message attack. We refer readers to [8] for a formal definition of authenticated symmetric-key encryption.

4 Threat Mode and System Design

A two-factor authentication scheme involves two players, User (U): an honest party which tries to prove its identity to a server by using a combination of a PIN and a device and Server (S): a semi-honest adversary which follows the protocol's instructions and tries to learn U's PIN. It also tries to authenticate itself to U.

We let the server communicate with the device through the user's computer, *i.e.*, the client. Specifically, similar to previous works (*e.g.*, those in [7,9,11]), we assume the device has a camera that lets the device scan a 2-D barcode containing messages the server sends to it via the client. Each of the above parties may have several instances running concurrently. In this work, we denote instances of user and server by U^i and S^j respectively. Each instance is called an oracle.

Figure 1 outlines the message flow of our 2FA scheme during the authentication phase. At a high level, the authentication phase works as follows. Any time the user wants to authenticate itself to the server, the user sends a message to the server via the client. The server replies to the client with a challenge and transaction details (Step 1). The user scans with the device the message that the client received (Step 2). The device shows the transaction details on the screen (Step 3). The user types the PIN into the device (Step 4). The device generates a response (Step 5). The user manually types the response into the client (Step 6). The client sends the response to the server for authentication (Step 7).

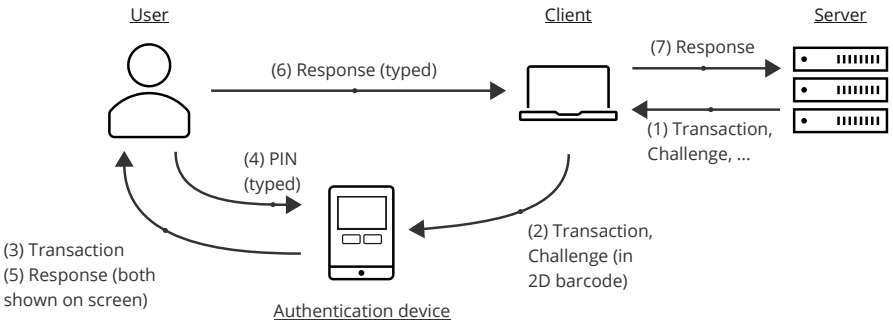

Fig. 1: Protocol participants and message flows

To formally capture the capabilities of an adversary, $\mathcal{A}$, in a hardware token-based 2FA, we mainly use the (adjusted) model proposed by Bellare *et al.* [3]. In this model, the adversary's capabilities are cast via queries that it sends to different oracles, *i.e.*, instances of the honest parties; the user and server interact with each other for some fixed number of flows, until both instances have terminated. By that time, each instance should have accepted holding a particular session key (sk), session id (SID), and partner id (PID). At any point in time, an oracle may *"accept"*. When an oracle accepts, it holds sk, SID, and PID. A user instance and a server instance can accept at most once. The above

model was initially proposed for password-based key exchange schemes in which the adversary does not corrupt either player. Later, Wang *et al.* [15] added more queries to the model of Bellare *et al.* to make it suitable for two-factor authentication schemes. The added queries allow an adversary to learn either of the user's factors (*i.e.*, either PIN or secret parameters stored in the hardware token) or the server's secret parameters. Below, we restate the related queries.

- $\texttt{Execute}(U^i, S^j)$: this query captures **passive** attacks in which the adversary, $\mathcal{A}$, has access to the messages exchanged between U^i and S^j during the correct executions of a 2FA protocol, π.
- $\texttt{Reveal}(I)$: this query models the misuse of the session key sk by instance I. Adversary $\mathcal{A}$ can use this query if I holds a session key; in this case, upon receiving this query, sk is given to $\mathcal{A}$.
- $\texttt{Test}(I)$: this query models the semantic security of the session key. It is sent at most once by $\mathcal{A}$ if the attacked instance I is "fresh" (*i.e.*, in the current protocol execution I has accepted and neither it nor the other instance with the same *SID* was asked for a Reveal query). This query is answered as follows. Upon receiving the query, a coin b is flipped. If $b = 1$, then session key sk is given to $\mathcal{A}$; otherwise (if $b = 0$), a random value is given to $\mathcal{A}$.
- $\texttt{Send}(I, m)$: this query models **active** attacks where $\mathcal{A}$ sends a message, m, to instance I which follows π's instruction, generates a response, and sends the response back to $\mathcal{A}$. Query $\texttt{Send}(U^i, \text{start})$ initialises π; when it is sent, $\mathcal{A}$ would receive the message that the user would send to the server.
- $\texttt{Corrupt}(I, a)$: this query models the adversary's capability to corrupt the involved parties.
  - if $I = U$: it can learn (only) one of the factors of U. Specifically,
    - if $a = 1$, it outputs U's PIN.
    - if $a = 2$, it outputs all parameters stored in the hardware token.
  - if $I = S$, it outputs all parameters stored in S.

Authenticated Key Exchange (AKE) Security. Security notions (*i.e.*, session key's semantic security and authentication) are defined with regard to the executing of protocol π, in the presence of $\mathcal{A}$. To this end, a game $Game^{ake}(\mathcal{A}, \pi)$ is initialized by drawing a PIN from the PIN's universe, providing coin tosses to $\mathcal{A}$ as well as to the oracles, and then running the adversary by letting it ask a polynomial number of queries defined above. At the end of the game, $\mathcal{A}$ outputs its guess b' for bit b involved in the Test-query.

Semantic security. It requires that the privacy of a session key be preserved in the presence of $\mathcal{A}$, which has access to the above queries. We say that $\mathcal{A}$ wins if it manages to correctly guess bit b in the Test-query, *i.e.*, manages to output $b' = b$. We denote its advantage as the probability that $\mathcal{A}$ can correctly guess the value of b; specifically, such an advantage is defined as $Adv^{ss}_{\pi}(\mathcal{A}) = 2Pr[b = b'] - 1$, where the probability space is over all the random coins of the adversary and all the oracles. Protocol π is said to be semantically secure if $\mathcal{A}$'s advantage is negligible in the security parameter, *i.e.*, $Adv^{ss}_{\pi}(\mathcal{A}) \leq \mu(\lambda)$.

Authentication. It requires that $\mathcal{A}$ must not be able: (a) to impersonate U, even if it has access to the traffic between the two parties as well as having access to either U's PIN, or its authentication device, or (b) to impersonate S, even if it has access to the traffic between the two parties. We say that $\mathcal{A}$ violates mutual authentication if some oracle accepts a session key and terminates, but has no partner oracle, which shares the same key. Protocol π is said to achieve mutual authentication if for any adversary $\mathcal{A}$ interacting with the parties, there exists a negligible function $\mu(.)$ such that for any security parameter λ the advantage of $\mathcal{A}$ (*i.e.*, the probability of successfully impersonating a party) is negligible in the security parameter, *i.e.*, $Adv_{\pi}^{aut}(\mathcal{A}) \leq \mu(\lambda)$.

In certain schemes (including ours), during the key agreement and authentication phase, the user needs to also verify a message, *e.g.*, a bank transaction. To allow such verification to be carried out deterministically, which will be particularly useful in the scheme's proof, we define a predicate $y \leftarrow \phi(m, \gamma)$, where $y \in \{0, 1\}$. This predicate takes as input a message m (*e.g.*, bank's transactions) and a policy γ (*e.g.*, a user's policy specifying a payment amount and destination account number). It checks if the message matches the policy. If they match, it outputs 1; otherwise, it outputs 0.

5 The Protocol

Recall that we wish to build an authentication protocol for which the server can verify that the PIN has been entered correctly but that an adversary cannot discover the correct PIN given access to challenge/response pairs and all data stored on the device, or access to all data stored on the server. These properties must be assured even when the PIN is small enough to be brute-forced. We achieve this goal through (a) performing the PIN verification only on the server, which imposes a rate limit on verification, (b) encrypting every sensitive message exchanged between the server and user using key-evolving symmetric-key encryption (*i.e.*, a combination of forward-secure pseudorandom bit generator and authenticated encryption), and (c) protecting against server compromise, by never directly sending the PIN to the server.

Our protocol consists of three phases: (i) a setup phase, performed once when the authentication device is manufactured, (ii) an enrolment phase for setting or changing a user's PIN, and (iii) an authentication phase in which the actual authentication is performed. As we already stated, each party has a unique (public) ID. We assume the parties include their IDs in their outgoing messages. Similar to other two-factor authentication schemes, we assume the server maintains a local threshold, and if the number of incorrect responses from a user within a fixed time exceeds the threshold, then the user and its device will be locked out. Such a check is implicit in the protocol's description.

5.1 Setup Phase

To bootstrap the protocol, in the setup phase, we require that the user and server share an *initial* randomly generated key k for AE and key st_0 for FS-PRG. The counter for the FS-PRG state is set to 0 on both sides. These values could be securely loaded into the device at the time of manufacture or can be sent (via a

secure channel) to the user who can use the device camera to scan and store them in the device. In this phase, the device generates and locally stores a random secret key sa^U for PRF. Figure 2 presents the setup in detail.

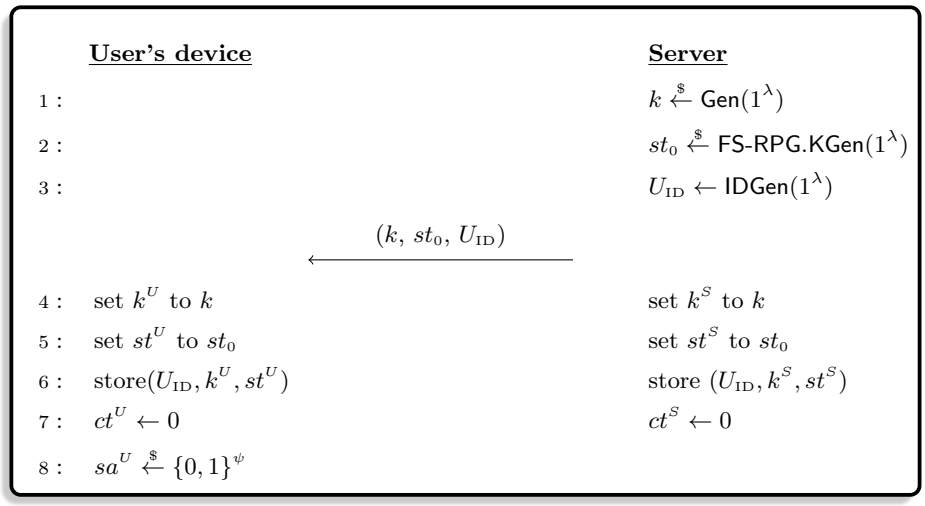

Fig. 2: Setup phase.

5.2 Enrolment Phase

The goal of the enrolment phase is to set the user's PIN, without providing the server with sufficient information to discover this PIN. At the end of this phase, the server will have stored the verifier v corresponding to the user's selected PIN. The steps involved in this phase are detailed in Figure 3.

We briefly explain how this phase works. The server first updates the FS-PRG's state, which results in a new state and random value kt_1^S; it also increments its counter by one. Then, the server generates a random challenge N^S. The server sends the enrolment challenge message which is a combination of the current counter and the challenge encrypted via the AE under the shared key k. On receiving this message, the client passes it to the device. The device decrypts the message using k that was shared with the server during the setup phase. If decryption succeeds, it extracts the server's challenge and counters from the message. To recover kt_1^S from the message the device's counter must be less than or equal to the counter it received from the server, which the protocol ensures is the case with a high probability (see Appendix A). Next, the device requests the PIN from the user and ensures it is what the user intends, *e.g.*, by requesting it twice and checking they match.

The device then generates a verifier v^U, by deriving a pseudorandom value from the PIN using PRF and the random key sa^U it generated in the setup phase. After that, the device locally synchronises the FS-PRG's state with the server

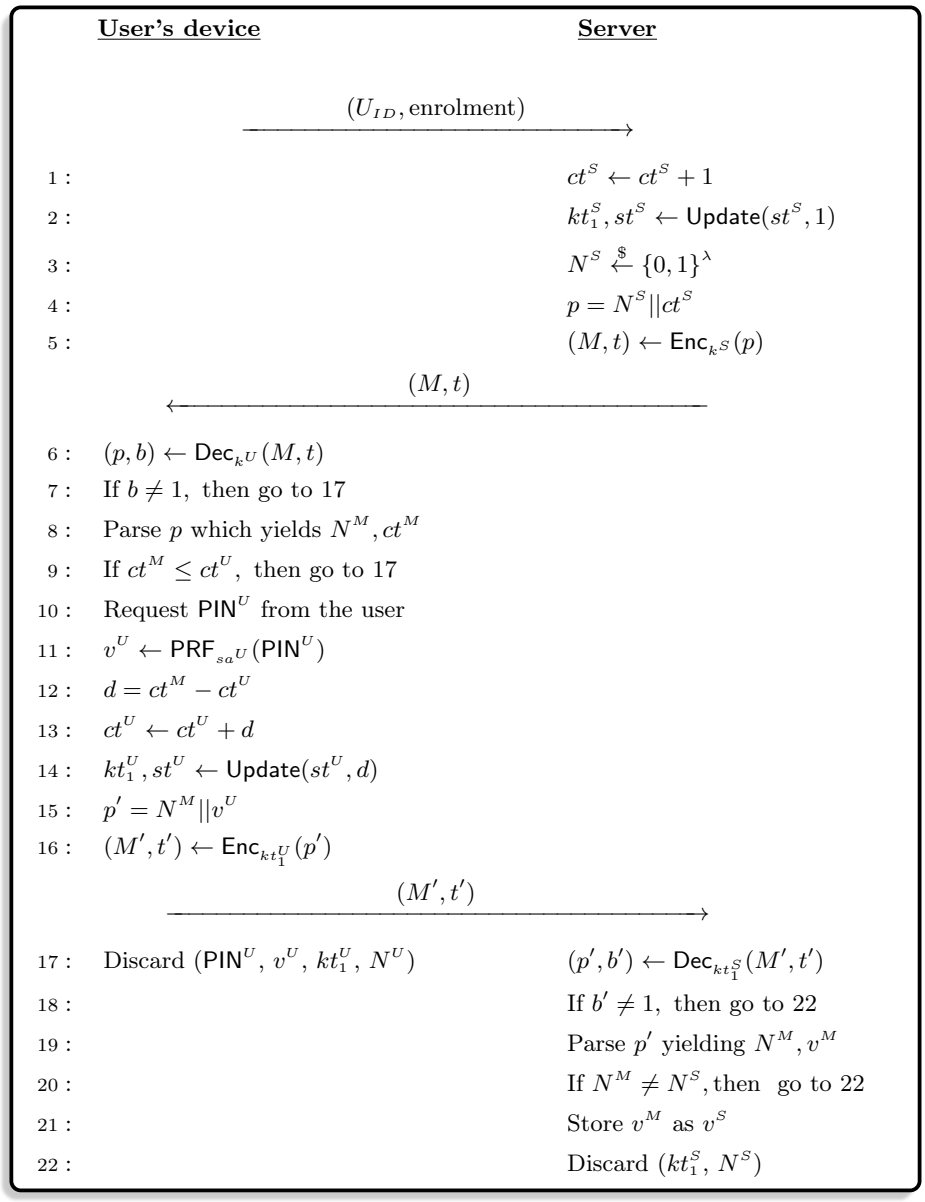

Fig. 3: Enrolment phase.

by updating the state until it matches the counter received from the server; this yields kt_1^U. This synchronisation is possible because the check at line 9 has already assured that the device's state is behind the server's state by at least one step. After the update, kt_1^U will equal kt_1^S because the initial FS-PRG's state is the

same (from the setup phase) and the two generators have been updated the same number of times. The user's device then encrypts the verifier and challenge under kt_1^U and sends this to the server. On receiving and validating this message, the server decrypts the message using $kt_1{}^S$, then extracts the challenge and verifier. If the challenge does not match the one corresponding to the current protocol exchange, the protocol halts. If the challenge does match, the server stores the verifier, v^S, associated with the user's account.

Finally, the device discards the challenge, kt_1^U, PIN, and v^U so that the PIN can no longer be recovered from the device. Note that the device can re-generate v^U using sa^U when the user types in their PIN again. The server also discards the challenge and $kt_1{}^U$ as they are no longer needed. Following the successful completion of this phase, the server stores the verifier corresponding to the user's PIN and the server and device will have synchronised their FS-PRG's state.

5.3 Authentication Phase

The goal of the authentication process is to give the server assurance that the device is currently present, the correct PIN has been entered, and the user has been shown the transaction that the server wishes to execute.

This phase works as follows. The server first updates the FS-PRG's state and corresponding counter, which results in a new state st^S, a new random value kt_2^S, and a new temporary counter tmp_{ct^S}. The server updates the state and the counter one more time which yields a new state st^S, a new random value kt_3^S, and a new counter ct^S. The server generates a random challenge and two ciphertexts, $\ddot{M}$ and $\hat{M}$. The former ciphertext consists of the random challenge and the description of the transaction, encrypted under key kt_2^S. The latter ciphertext contains the counter tmp_{ct^S}, encrypted under key k^U. The reason tmp_{ct^S} is encrypted under key k^U is to allow the device to decrypt the ciphertext easily in case of previous message loss; for instance, when the server sends $(\ddot{M}, \hat{M})$ to the server, but they are lost in transit, multiple times, and a fresh pair finally arrives at the client after the server sends them upon the user's request. Encrypting tmp_{ct^S} under key k^U (instead of one of the evolving keys) lets the device deal with such a situation.

Upon receiving the ciphertexts, the device validates and decrypts the messages. It extracts the challenge N^M, counter tmp_{ct^S}, and transaction t^M. It ensures that its own counter is behind the received counter. As will be discussed in Appendix A, this check will succeed with high probability. The device synchronises its state and counter using the server's messages. Next, the device displays the transaction for the user to check. If the user does not accept the transaction (*e.g.*, due to an attempted man-in-the-browser attack), then the protocol aborts immediately. Assuming the user is willing to proceed, then the device prompts for the PIN, and computes the verifier v^U using the key sa^U. If the user enters the correct PIN, the verifier will be the same as the one sent to the server during the enrolment phase.

For the device to generate the response message, it first updates its state one more time, which results in a pseudorandom value kt_3^U. Then, it derives a pseudorandom value, $response^U$, from a combination of the random challenge

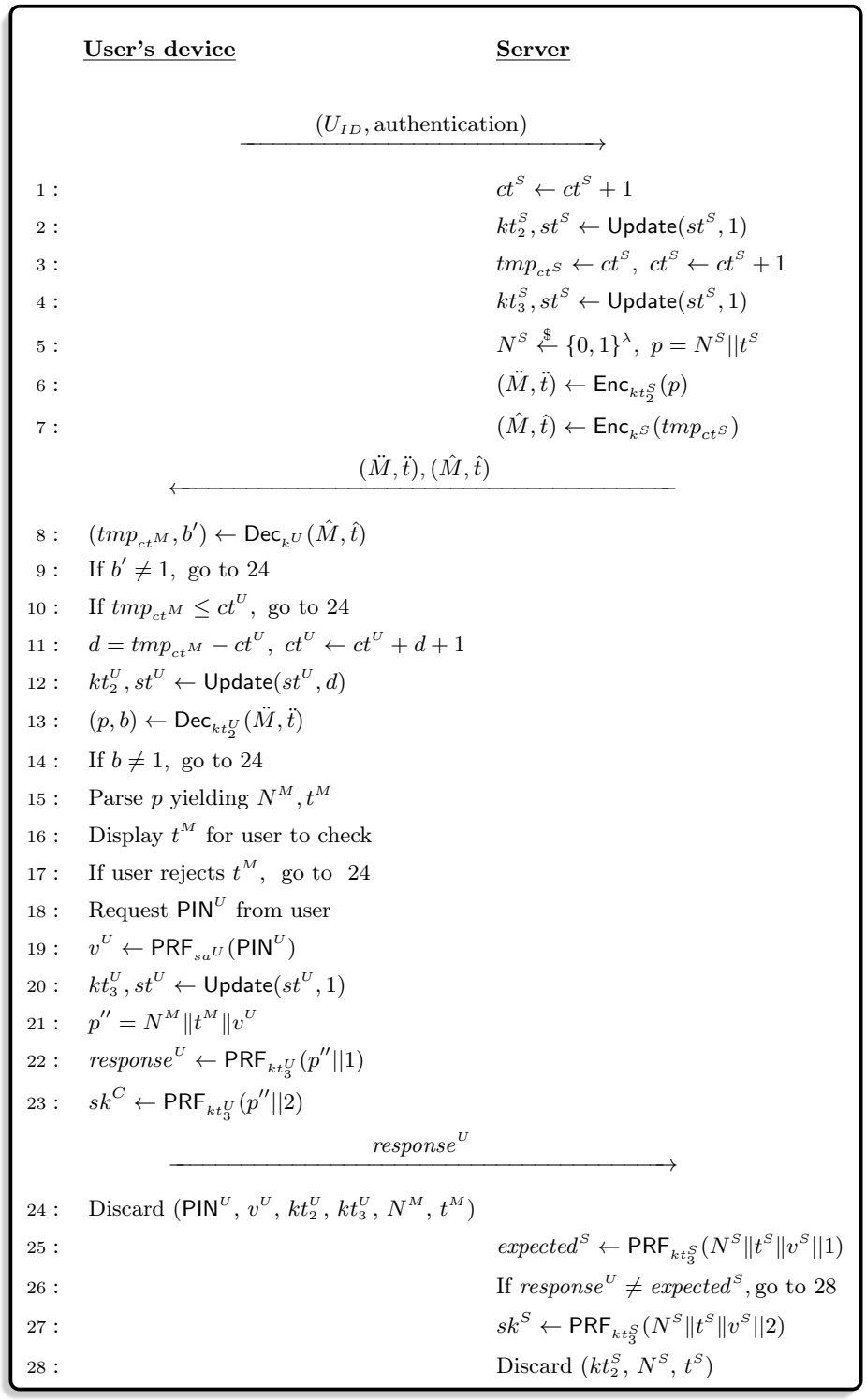

Fig. 4: Authentication phase.

N^M, transaction t^M, verifier v^U, and $x = 1$ using PRF and kt_3^U. The device generates a session key, using the above combination and key with a difference that now $x = 2$. The response message is truncated to be a convenient length, displayed on the screen of the device, typed by the user into the client, and sent to the server. The device discards the PIN, the verifier, all FS-PRG keys, the challenge, and the transaction's description, so as to protect the PIN from discovery. The server computes the expected response message based on its own values of the challenge, transaction, and verifier. Note that the verifier is retrieved from the value set during the enrolment phase. The server then compares the expected response with the response sent by the client. Only if they match, the authentication is considered to have succeeded. If the response does not match the one the server expects this could indicate that the message was tampered with, or that the user entered an incorrect PIN. Next, the server generates the session key in the same way as the device does. The server also discards the FS-PRG key, the challenge, and the transaction's description.

Below, we formally state the security of our protocol. First, we present a theorem stating that the advantage of an adversary in breaking the semantic security of the above protocol is negligible.

Theorem 1 (Semantic Security). *Let $\mathcal{A}$ be a probabilistic polynomial time (PPT) adversary with less than q_s interactions with the parties and q_p passive eavesdropping, i.e., number of local executions. Let λ be a security parameter and $Adv_{\pi}^{ss}(\mathcal{A})$ be $\mathcal{A}$'s advantage (in breaking the semantic security of an AKE scheme π) as defined in Section 4. Then, such an advantage for the protocol ψ has the following upper bound:*

$$Adv_{\psi}^{ss}(\mathcal{A}) \leq 2(q_s + q_p)\Big(Adv^{\mathsf{PRF}}(\mathcal{A}) + Adv^{Enc}(\mathcal{A})\Big) + \frac{8(2q_s + q_p)}{2^{\lambda}}$$

Next, we present a theorem stating that the advantage of an adversary in breaking the authentication of the above protocol is negligible.

Theorem 2 (Authentication). *Let PIN be an element distributed uniformly at random over a finite dictionary of size N. Also, let $\mathcal{A}$ be a PPT adversary with less than q_s interactions with the parties and q_p passive eavesdroppings. Let λ be a security parameter and $Adv_{\pi}^{aut}(\mathcal{A})$ be $\mathcal{A}$'s advantage (in breaking the authentication of an AKE scheme π) as defined in Section 4. Then, in the protocol ψ, $Adv_{\psi}^{aut}(\mathcal{A})$ has the following upper bound:*

$$Adv_{\psi}^{aut}(\mathcal{A}) \leq (q_s + q_p)\Big(Adv^{\mathsf{PRF}}(\mathcal{A}) + Adv^{Enc}(\mathcal{A})\Big) + \frac{9q_s + 4q_p}{2^{\lambda}} + \frac{q_s}{N}$$

We refer to Appendix B for informal security analysis and the paper's full version [2] for formal proofs of the above two theorems.

6 System Usability

Usability is of critical importance for an effective authentication system as otherwise, users will refuse to use it or implement insecure workarounds [5]. In our protocol, the server interacts with the device via the client. To accommodate usability and let the device receive the server's message, we require the device to be

able to receive a few hundred bytes from the server. This functionality is already present on any device capable of transaction authentication because it must be able to receive a description of the transaction to show to the user. Typically this communication functionality is implemented by an inexpensive camera such as in the Gemalto SWYS QR [11] or OneSpan Digipass 770 [9], which both scan a 2D barcode shown on the screen of the client. The response from the device to the server can be safely truncated because the protocol ensures offline brute-force attacks are not possible. So, the response can be manually typed without any special hardware required for this direction of communication. The response length should be selected to reduce the chance of success of an online brute force attack to an acceptable level, taking into consideration the rate-limiting implemented on the server. This security-usability trade-off is not specific to our protocol and exists in all hardware token-based multi-factor authentication schemes that do not assume a high-bandwidth communication channel from the authentication device to the server.

Another consideration is handling mistyped or forgotten PINs. As we highlighted in Section 5, when setting the PIN, the device can ask the user to confirm their PIN by entering the PIN twice and alerting the user if they do not match. Because we assume that the device has no trusted hardware we cannot store the PIN in the device. So, during authentication, if a wrong PIN is entered, the user will only be alerted after the response code is verified by the server. To enhance usability by detecting mistyped PINs earlier in the protocol, at some cost of security, the device could show the user an image computed as a function of the PIN entered to help the user detect a mistyped PIN, serving as a checksum. To prevent someone observing the device from discovering the PIN from the image, the function could be designed to have a large number of collisions.

7 Evaluation

In this section, we briefly analyse and compare our 2FA protocol with the smart-card-based protocol proposed in [15] and the hardware token-based protocol in [7] as the latter two protocols are relatively efficient, do not use secure chipsets, and they consider the same security threats as we do. We summarise the analysis result in Table 2. We refer readers to the full version for a more detailed evaluation.

Table 2: Comparison of efficient two-factor authentication protocols.

Features	Operation	Our Protocol	[15]	[7]
Computation cost	Sym-key	18	19	7
	Modular expo.	0	5	12
Communication cost	—	2804-bit	3136-bit	3900-bit
Not requiring multiple pass/PIN	—	✓	×	×
Not requiring modular expo.	—	✓	×	×
Security assumption	—	Standard	Random oracle	Random oracle

7.1 Computation Cost

In our protocol, each party (user or server) invokes the authenticated encryption scheme 4 times and the pseudorandom function 5 times. In the protocol proposed by Wang *et al.* [15], the user invokes a hash function 11 times and performs 3 modular exponentiations while the server invokes the hash function 8 times and performs 2 modular exponentiations. Moreover, in the protocol presented by Jarecki *et al.* [7], the user invokes a hash function 3 times and runs symmetric key encryption once. It also performs 10 modular exponentiations. While the server invokes a pseudorandom function once and performs at least 2 modular exponentiations and 2 symmetric-key encryptions. Thus, our protocol and the ones in [15,7] involve a constant number of symmetric key primitive invocations; however, our protocol does not involve any modular exponentiations, whereas those in [15,7] involve a constant number of modular exponentiations which leads to a higher cost. The avoidance of the need to perform modular exponentiations not only reduces computation cost and energy consumption but also saves flash storage due to not requiring a big integer library.

7.2 Communication Cost

In our protocol, the communication cost of the user is 1268 bits and the server is 1536 bits. However, in the protocol proposed in [15], the communication cost of the user is 1952 bits and the server is 1184 bits; while in the protocol developed in [7] the user's and server's communication costs are at least 2856 and 1044 bits respectively. Hence, our protocol imposes 10% and 40% lower communication costs than the protocols in [15] and [7] do respectively.

7.3 Other Features

In our protocol, a user needs to know and type into the device only a single secret, *i.e.*, a PIN. In contrast, in the protocol in [15], a user has to know and type an additional secret, *i.e.*, a random ID. As shown by Scott [10], the scheme in [15] will not remain secure, even if only the user's ID is revealed. The protocol in [7] requires the user (in addition to remembering its PIN) to locally store a cryptographic secret key of sufficient length, *e.g.*, 128 bits; this secret key must not be kept on the device but must be typed into it during the authentication phase. Furthermore, our protocol is secure in the standard model whereas the protocols in [15,7] are in the non-standard random oracle model.

8 Conclusion and Future Work

We have presented a 2FA protocol that resists a strong adversary who may (a) observe the traffic between a user and server, and (b) have physical access to the user's device, or its PIN, or breach the server. Our protocol offers a unique combination of key features that state-of-the-art schemes do not. Specifically, our protocol (i) requires a user to remember only one secret/PIN, (ii) is based on only symmetric key primitives, (iii) is in a standard model, and (iv) imposes low communication costs. This is the first protocol that offers the aforementioned features without requiring tamper-resistant hardware. Future research could investigate the usability of a hardware token that embeds our 2FA protocol.

References

1. Payment Services (PSD 2). Directive 2015/2366/EU of the European Parliament and of the Council (November 2015)
2. Anonymous: A forward-secure efficient two-factor authentication protocol–Full Version (2023), https://github.com/ESORICS2020/full-version/blob/main/main.pdf
3. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: EUROCRYPT (2000)
4. Bellare, M., Yee, B.S.: Forward-security in private-key cryptography. In: CT-RSA (2003)
5. De Cristofaro, E., Du, H., Freudiger, J., Norcie, G.: A comparative usability study of two-factor authentication. arXiv preprint arXiv:1309.5344 (2013)
6. Executive Office of the President: Office of Management and Budget: Moving the U.S. government toward zero trust cybersecurity principles (2022), https://www.whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf
7. Jarecki, S., Jubur, M., Krawczyk, H., Saxena, N., Shirvanian, M.: Two-factor password-authenticated key exchange with end-to-end security. ACM Trans. Priv. Secur. (2021)
8. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. CRC Press (2014)
9. OneSpan: Digipass 770 datasheet, https://www.onespan.com/resources/digipass-770/datasheet
10. Scott, M.: Cryptanalysis of a recent two factor authentication scheme. IACR Cryptol. ePrint Arch. p. 527 (2012)
11. Thales: Gemalto SWYS QR Reader Eco (2020), https://www.thalesgroup.com/sites/default/files/database/document/2020-12/fs-QR-code-reader.pdf
12. View, M., M'Raihi, D., Hoornaert, F., Naccache, D., Bellare, M., Ranen, O.: HOTP: An HMAC-Based One-Time Password Algorithm. RFC 4226 (Dec 2005)
13. View, M., Naccache, D., Rydell, J., Bajaj, S., Machani, S.: OCRA: OATH Challenge-Response Algorithm. RFC 6287 (Jun 2011)
14. View, M., Rydell, J., Pei, M., Machani, S.: TOTP: Time-Based One-Time Password Algorithm. RFC 6238 (May 2011)
15. Wang, D., Wang, P.: Two birds with one stone: Two-factor authentication with security beyond conventional bound. IEEE Trans. Dependable Secur. Comput. (2018)

A Synchronisation

A user's device needs to be synchronised with the server in order for the server to check the correctness of the response generated by the device. This is particularly the case in our proposed protocol because if one side advances too far, it is by design impossible for it to move backwards. Specifically, we must provide assurance that the server state remains at the same state as the device's state, or that the server is ahead of the device, *i.e.*, $ct^S \geq ct^U$. Then, as challenge messages always contain the current value of the server's counter, the device is always able to catch up with the server. We achieve this via three approaches. Firstly, by requiring the FS-PRG's state to advance with the counter, such that the counter is consistent with the state. Secondly, by requiring that the device never advances its state directly, but only advances to the point that the server

currently is at. Thirdly, by requiring the device only to advance its state in response to an authenticated challenge from the server.

The protocol takes into account the case where messages are dropped. Response messages are not involved in advancing the forward-secure state; therefore, if these messages are dropped, then it would not have any effect on synchronisation. However, challenge messages are important, if any of them is dropped, then the device would not advance the state and would be behind the server. Nevertheless, this would not cause any issues, because the server's next challenge message will include the new value of the counter and the device will advance the state until it matches the server's state. Hence, in the authentication phase (in Figure 4), inequality $ct^S \geq ct^U$ must hold in the following three cases:

- **Case 1: no messages are dropped.** This is a trivial case and boils down to the correctness and security of the authentication protocol (presented in Figure 4). Specifically, in this case, inequality $ct^S \geq ct^U$ always holds because the device advances its state only after it receives a valid challenge from the server that has already advanced its state. An adversary would be able to convince the device to advance its state (before the server does so) if it could generate a valid encrypted challenge; however, its probability of success is negligible in the security parameter, due to the security of Authenticated Encryption (AE).
- **Case 2: server challenges are dropped.** If the adversary drops the server's encrypted challenge message $(\ddot{M}, \ddot{t})$, then the device would not advance its state. Because the device advances its state (at line 12) only after it receives the encrypted messages, checks their validity and makes sure its state is smaller than the server's state. The server advanced its state, regardless of whether its message will be dropped. Thus, inequality $ct^S \geq ct^U$ always holds in Case 2 as well.
- **Case 3: user responses are dropped.** If the adversary drops the user's response (*i.e.* $response^U$), then the user cannot authenticate itself to the server, but it can re-execute the authentication protocol, by sending to the server message $(U_{ID}, \text{authentication})$. In Case 3, both the device and server have updated their state before the message is dropped; so, inequality $ct^S \geq ct^U$ holds in this case.

Note that the FS-PRG advance process is fast; thus, multiple invocations of this will not create a noticeable delay. Note that in the case where the enrolment's response message is dropped, the PIN will remain unchanged; as a result, the user may be surprised that the new PIN does not work. But, the old PIN will keep working and enrolment can be repeated to update the PIN.

B Informal Security Analysis

In this section, we informally analyse the security of the proposed protocol. We refer readers to the paper's full version [2] for formal proof. We analyse its security through five scenarios defined in terms of adversary capabilities and protection goals. The scenarios are designed to assume a strong adversary so that the results are generalisable to other situations, but are constrained so as to make sense, e.g., we assume that at least one factor is secure.

B.1 Threats and Protection objectives

In this section, we first list a set of threats that our protocol must resist.

- *T.DEV: Device access.* An adversary may steal the authentication device. The adversary will then know k^U, sa^U and the current values of ct^U and st^U, because we assume the device does not take advantage of a trusted chipset. The adversary does not learn v^U, t^M, PIN^U, kt_1^U, kt_2^U, kt_3^U or previous values of st^U; as these are all discarded at the end of a protocol exchange. We assume that the user will not use the device after it has been stolen, and will be issued with a replacement.
- *T.MITM: Man-in-the-middle.* An adversary may have access to the traffic exchanged between the client and server.
- *T.PIN: Knowledge of PIN.* An adversary may know the PIN entered by a user, for example from observing them type it in.
- *T.SRV: Server compromise.* The server is the party relying on the authentication, so it does not make sense for the server to be wholly malicious, *i.e.*, an active adversary. However, it is reasonable to believe that the server database could be compromised, disclosing k^S, v^S, and the current values of ct^U and st^U.

Next, we present the security objective that our protocol must achieve.

- *O.AUTH: Authentication.* If the server considers the authentication to have succeeded then the correct device was used and the correct PIN was entered.
- *O.TRAN: Transaction authentication.* If the server considers the authentication to have succeeded then the correct device was used, the correct PIN was entered, and the device showed the correct transaction.
- *O.PIN: PIN protection.* The adversary should not be able to discover the user's PIN.

B.2 Scenarios

In this section, we briefly explain why the protocol meets its objective in different threat scenarios.

1. *O.AUTH against T.PIN and T.MITM.* The first scenario we consider is the case where the adversary does not have access to the authentication device but does know the user's PIN and communication between the client and server. For the adversary to perform a successful authentication, it must compute $\mathsf{PRF}_{kt_3^U}(N^M \| t^M \| v^U || 1)$. Nevertheless, it does not know kt_3^U or the state from which kt_3^U has been generated; since kt_3^U is an output of PRF and is sufficiently large, it is computationally indistinguishable from a truly random value. The probability of finding it is negligible in the security parameter. Thus, the only party which will generate a valid response is the device itself (when the PIN is provided) at line 22 of Figure 4. We have already assumed that the adversary does not have access to the device; therefore, it cannot generate a valid response.

2. *O.AUTH against T.DEV and T.MITM.* In this scenario, the adversary has compromised the user's device (but not its PIN), has records of previous messages, and wishes to impersonate the user. Since the random challenge in the expected response is unique, and the PRF provides an unpredictable output, previous responses will not be valid; so, a reply attack would not work. The adversary can use the device to discover (kt_2^U, kt_3^U) and all the parameters of the response message, except the PIN. In this case, it has to perform an online dictionary attack by guessing a PIN, using the extracted parameters to generate a response, and sending the response to the server. But, the server will lock out the device if the number of incorrect guesses exceeds the predefined threshold. Other places where the PIN is used are in (i) the enrolment response, where the verifier derived from the PIN is encrypted under an evolving fresh secret key, and (ii) the authentication response, where the response is a pseudorandom value derived from the PIN's verifier using an evolving fresh secret key. In both cases, the evolving keys cannot be obtained from the current state, due to the security of FS-PRG.
3. *O.PIN against T.DEV and T.MITM.* The adversary has compromised the device and wishes to obtain the user's PIN. As with Scenario 2, the PIN cannot be obtained from the device, the responses in the authentication or enrolment phases.
4. *O.PIN against T.SRV and T.MITM.* The adversary has compromised the server and wishes to obtain the user's PIN. In this case, the adversary has learned the verifier but does not know the value of the secret key, used to generate the verifier. If the server retains values of the verifier for previous PINs (in the case where the server does not delete them), then the adversary would also learn further verifiers for the same device. The PIN is only used for computing the verifier, so the only way to obtain the PIN would be to find the key of the PRF which is not possible except for a negligible probability in the security parameter. The only information this discloses is that if two values for the verifier are equal, then that implies that two PINs for the same device were equal. Even this minimal information leakage can be removed if the server rejects the PINs that were used before.
5. *O.TRAN against T.PIN and T.MITM or T.DEV and T.MITM.* As we discussed above, an adversary cannot successfully authenticate, even if it sees the traffic between the client and server and has access to either the PIN or the device. Furthermore, due to the security of the authenticated encryption, the device can detect (except for a negligible probability) if the transaction's description, that the server sends to it, has been tampered with.

B.3 Excluded Scenarios

We exclude some scenarios that do not make sense or are not possible to secure against.

- *Compromised PIN and device.* If the adversary has compromised both factors of a 2FA solution, then the server cannot distinguish between the adversary and the legitimate user.

- *Authentication on server compromise.* If the adversary has compromised the server, then it can either directly perform actions of the server or change keys to ones known by the adversary. Therefore, it does not make sense to aim for O.AUTH in this situation.
- *Compromised server and device.* If the adversary has compromised the server and the device, then the PIN can be trivially brute-forced with knowledge of v^S and sa^U.