Provable Security Analysis of the Secure Remote Password Protocol

Dennis Dayanikli and Anja Lehmann

Hasso-Plattner-Institute, University of Potsdam
{dennis.dayanikli, anja.lehmann}@hpi.de

Abstract. This paper analyses the Secure Remote Password Protocol (SRP) in the context of provable security. SRP is an asymmetric Password-Authenticated Key Exchange (aPAKE) protocol introduced in 1998. It allows a client to establish a shared cryptographic key with a server based on a password of potentially low entropy. Although the protocol was part of several standardization efforts, and is deployed in numerous commercial applications such as Apple Homekit, 1Password or Telegram, it still lacks a formal proof of security. This is mainly due to some of the protocol's design choices which were implemented to circumvent patent issues.
Our paper gives the first security analysis of SRP in the universal composability (UC) framework. We show that SRP is UC-secure against passive eavesdropping attacks under the standard CDH assumption in the random oracle model. We then highlight a major protocol change designed to thwart active attacks and propose a new assumption – the additive Simultaneous Diffie Hellman (aSDH) assumption – under which we can guarantee security in the presence of an active attacker. Using this new assumption as well as the Gap CDH assumption, we prove security of the SRP protocol against active attacks. Our proof is in the "Angel-based UC framework", a relaxation of the UC framework which gives all parties access to an oracle with super-polynomial power. In our proof, we assume that all parties have access to a DDH oracle (limited to finite fields). We further discuss the plausibility of this assumption and which level of security can be shown without it.

1 Introduction

A password authenticated key-exchange protocol (PAKE) [13,17] allows two parties to securely establish a cryptographic session key over an insecure channel based on their knowledge of a shared low-entropy password. In its asymmetric version – aPAKE [15,22,36] – one of the parties plays the role of the client while the other party acts as the server. Upon registering a client, the server stores a mapping of the password (the password verifier) which is typically some form of a salted hash of the password. After registration, both parties can engage in a protocol to establish a common key. To do so, the server uses the password verifier while the client uses its password. A secure aPAKE protocol leaks no offline-attackable information about the password or the password verifier in the

key exchange phase, and ensures that both parties agree on a mutually shared strong cryptographic key only if the passwords match. In the asymmetric setting, the protocol often also ensures explicit client authentication, meaning that the server learns whether the client used the correct password in the key exchange. This allows the server to detect whether a client account is subject to an online password guessing attack, and apply rate-limiting accordingly.

The idea of aPAKE was first introduced by Bellovin and Merritt [15], and the desirable security properties were subsequently formalized via game-based definitions [18], simulation-based definitions [22] and through an ideal functionality in the Universal Composability (UC) framework [36]. Despite the practical motivation and availability of efficient and simple protocols, most (a)PAKE schemes have not seen wide-spread adoption yet. This is particularly disappointing for asymmetric PAKE, which could serve as a significantly more secure replacement for classic password-based authentication where passwords are transmitted in plaintext to the server. As an attempt to push (a)PAKE towards real-world adoption, the Internet Engineering Task Force has recently run a selection process to choose (a)PAKE protocols for standardization [58]. This standardization effort has created renewed attention to the field of password-authenticated key exchange and also led to formal security proofs of previously proposed yet not formally analysed protocols [5,63].

However, the currently most widely deployed aPAKE protocol was not included in the selection process and still lacks a formal security analysis: the "Secure Remote Password" (SRP) protocol.

The SRP Protocol. One of the earliest aPAKE protocols is the SRP protocol developed in 1997 [2] and first described in a paper by Wu [73] in 1998. Wu introduced the protocol as an instantiation of Asymmetric Key Exchange (AKE) since at that time the term aPAKE had not been coined yet. SRP is distributed on Open Source-friendly terms and was specifically designed to avoid export control [73] and work around existing patents [2] in the area, at a time where existing PAKE protocols such as EKE [16] and SPEKE [44] were patented. This enabled it to become one of the most widely deployed aPAKE protocols to this date: SRP is used for client authentication in various commercial applications such as the password manager 1Password [35], the ING Diba InsideBusiness app [42], Telegram's 2FA protocol [68], SecureSafe's file manager and password manager [59], TeamViewer [67], Apple's iCloud keychain escrow [9] and HomeKit Accessory Protocol [10] that is deployed on more than a billion Apple devices, or the secure email system of Proton Mail [23]. In April 2023, Proton also announced that SRP is used for authentication in their newest product – the password manager Proton Pass [74]. Additionally, authentication protocols such as TLS 1.2 [66] or AWS's Amazon Cognito [56] allow incorporating the SRP protocol as a password-based authentication method.

The two most widely used versions of the SRP protocol are versions SRP-3 [73] and SRP-6a [71]. SRP-6a is the most recent version and addresses vulnerabilities of the SRP-3 protocol. The Stanford SRP homepage [2] also lists

two prior versions of the protocol (SRP-1 and SRP-2) whose vulnerabilities were already addressed by SRP-3.

High Level Description of the Protocol. The SRP protocol works in a finite field. Upon client registration, the server stores a password verifier $v = g^x$ using a primitive element g of the field and the salted hash value $x = H(s, pw)$ of the client's password pw. This password verifier is later used in the login phase, to enable the client and server to establish a strong shared key K if the client uses the same password as during registration.

On a high level, the login phase of the SRP protocol is a Diffie-Hellman key exchange with a twist of mixing the password into the public keys. In SRP, the server's Diffie-Hellman public key g^b is hidden by the password verifier v. Thus, the server's message is $B = v + g^b$ while the client's message is a regular Diffie-Hellman public key $A = g^a$. Both parties compute an ephemeral key[1] $T = g^{ab+bx}$. The server computes this value as $T = (Av)^b$, while the client computes $T = (B - g^x)^{a+x}$ where x is derived from the password attempt pw'. In order for the client to compute $x = H(s, pw')$, the server must send the salt s to the client. If the passwords match, both parties will compute the same value T which is then hashed into the shared key $K = H(T)$.

SRP-6a additionally ensures mutual explicit entity authentication [57] and key confirmation by incorporating one round of key confirmation messages into the protocol. At the end of a successful protocol run, both parties are convinced that the counterparty holds the same key and since the key can only be derived with a secret password (file) known by the counterparty, this yields explicit entity authentication.

Challenging Design Choices & Consequences. Since the SRP protocol uses both addition and multiplication in the finite field, it is not possible to instantiate it in a group. As a result, the protocol is deprived of the benefits provided by elliptic curve cryptography such as smaller key sizes and faster computation times compared to finite field cryptography [50]. Furthermore, the utilization of both finite field operations complicates the security analysis of the protocol, and this is likely the reason why a formal security proof for the SRP protocol has not been established yet.

Despite not being backed up through provable security, the SRP protocol was part of standardization processes by respected organizations such as the IETF (RFC2945 [72], RFC5054 [66]), IEEE (P1363.2 [1]) and ISO (11770-4 [43]) and continues to be widely used in commercial products.

However, SRP was not included in the latest IETF CFRG PAKE selection, due to the lack of a formal analysis and the incompatibility with elliptic curve cryptography. In this IETF process, the OPAQUE [47] protocol was chosen as the recommended aPAKE. OPAQUE has a formal security proof, can be implemented more efficiently since it does not depend on finite field arithmetic, and also provides stronger security as it is resistant to precomputation attacks.

[1] In SRP-6a this computation includes two scrambling parameters u and k but for the sake of this overview we ignore these parameters.

Thus, while currently better aPAKE protocols than SRP exist, we strongly believe that the real-world dominance of SRP to date demands a thorough investigation of its provable security guarantees.

UC Security. For password-based schemes, and (a)PAKE in particular, security in the UC framework [24] is considered the gold standard as it does not make any assumptions on password distributions, models real-world behaviour such as password re-use and guarantees secure composition with arbitrary other protocols. An ideal functionality for aPAKE protocols was introduced by Gentry et al. [36] and has been refined over the years to address initial oversights. Several aPAKE schemes, including [36], OPAQUE [47], AuCPace [38], KHAPE [37], SPAKE2+ [28,63] and OKAPE [34] have been formally proven to be secure aPAKE protocols in the UC framework.

1.1 Our Contributions

In our work, we give the first formal security analysis of the SRP protocol and prove the scheme secure in the UC framework.

We first show that both SRP-3 and SRP-6a achieve standard UC-security in the presence of a semi-passive attacker in the random oracle model [14] and under standard assumptions. The semi-passive setting of our model extends the classic eavesdropping attack by additionally allowing the adversary to steal the password file of the server.

We then provide a thorough discussion of the protocol changes made from SRP-3 to SRP-6a. In particular, we introduce an assumption under which a two-for-one-guessing attack – a malicious server being able to make two (or more) online password guesses in one protocol run [71] – is infeasible in the SRP-6a protocol variant. The new assumption is called additive simultaneous computational Diffie Hellman (aSDH) assumption and asks an adversary to simultaneously solve two CDH instances in a finite field where the two instances are linked by a random element.

Finally, we prove security of SRP-6a against active adversaries. This proof is in the random oracle model and under the (Gap) aSDH and Gap CDH assumption. Our proof additionally requires a helper oracle to be available throughout the simulation of the protocol to all parties, including the simulator. The required helper oracle is a DDH oracle which enables us to "extract" passwords from a malicious client. Roughly, the simulator can check if a maliciously provided T value is a valid DH value for any of the $x = H(s, pw')$ values the adversary has previously obtained from the random oracle. If so, the simulator uses the same pw' towards the functionality to learn if the password is correct and adjust its simulation accordingly.

Whereas a DDH oracle is a commonly made assumption used within a security reduction in the analysis of PAKE protocols [3,37], our assumption is different, as we rely on it as a helper oracle for the entire simulation. In the wider context of UC security proofs, this technique of relying on a (superpolynomial) helper oracle has been studied and used for relaxations as *angel-based*

UC security [26,52,54]. Such angel-based UC proofs maintain the composition guarantees if the helper oracle is globally available to all parties, i.e., including the real-world adversary. Offering a global DDH oracle might appear like a strong limitation then, but it is not – now thanks to the unusual design choice of the SRP protocol: The DDH oracle is limited to the finite field used by the SRP protocol where it is known that the DDH assumption does not hold. Thus, the presence of such an oracle has no impact on security if SRP is combined with other protocols that rely on the DDH assumption, as they of course will work in groups in which the assumption holds.

While we believe that relying on this helper oracle assumption is not a significant limitation, we also argue about partial security guarantees that can be made without it: if one assumes an honest-client setting, SRP-6a is UC-secure under the standard CDH assumption together with the (strong) aSDH assumption.

As a side contribution, we identify a shortcoming of the UC aPAKE security definition used by Jarecki et al. [37]. A secure aPAKE protocol according to this definition does not inform the adversary of a failed key exchange between the client and the server. In aPAKE protocols with explicit authentication where one party aborts the communication if the authentication fails (as in SRP or KHAPE [37]), this is not attainable. Our solution is to reintroduce a feature which was already incorporated in the security definition of Hesse [41] and which allows to prove security of these protocols.

1.2 Previous Work on the Security of SRP

In [73], Wu argues that SRP-3 is secure against passive eavesdropping attacks and resistant to offline attacks first described by Denning and Sacco [32]. In this type of attack, an adversary eavesdrops a session key and uses this either to impersonate a client or to recover the password via an offline brute force attack. We argue that his proof of security is flawed, as the reduction assumes a specific behaviour of the adversary. However, this can be fixed easily, and our proof of UC security in the semi-passive case confirms his claim that the protocol is secure against passive eavesdropping attacks. Our proof even shows stronger security as we consider a semi-passive adversary who is allowed to additionally steal the server's password file.

In [71], Wu shows a two-for-one guessing attack on SRP-3. He proposes SRP-6 which mitigates this attack but does not give a proof on how this change prevents the attack. We propose an assumption under which the two-for-one-guessing attack is infeasible and use it to prove security of the SRP-6a protocol which was presented shortly after SRP-6 [2].

Sherman et al. [61] provided a security analysis of SRP-3 using the Cryptographic Protocol Shapes Analyzer, a software tool designed to assist in the design and analysis of cryptographic protocols. Since the software tool does not allow the modeling of both modular addition and multiplication in the finite field, they modeled the hiding of v to $v + g^b$ as an encryption. This is an idealized assumption and does not allow to find attacks which use the algebraic structure of the finite field. They found that the SRP-3 protocol does not leak

any secrets to eavesdropping attackers. Our work confirms their computer-aided proof in a theoretic framework. Additionally, their analysis identified a property of the protocol which allows a malicious server to fake an authentication session with a client using only the compromised password verifier and not the client's password. While the paper highlights this as a notable weakness, it is actually a security feature of the SRP protocol. Similar as in the simulation arguments for zero-knowledge proofs, the ability of a malicious server to simulate the authentication protocol with a client without knowing the client's password guarantees that no information about the password can be deduced from the protocol.

De Almeida Braga et al. [31] showed that implementations of SRP are vulnerable to offline dictionary attacks using side-channel timing attacks. We emphasize that care needs to be taken when implementing the SRP protocol but the analysis of side-channel attacks is outside the scope of our work.

Hao [39] showed that every protocol run in SRP-6 leaks some knowledge of the password verifier to an attacker. He exploited the algebraic structure of the client's ephemeral key $T = (B - g^x)^{a+x}$ (again, we omit scrambling parameters k and u here) which might fall into a smaller subgroup of the finite field. If it does not fall into a smaller subgroup, a malicious server (who has not compromised the database) can rule out four password verifiers in one protocol execution with the client. While this is a neat attack which shows another side of the small-subgroup attack, the extra knowledge gained by the adversary is negligible since the password verifiers are random field elements of a large field and derived from a hash function of the password.

Roadmap. We describe the standard aPAKE security model and our modifications in Section 2 and hardness assumptions in Section 3. The SRP-6a protocol is detailed in Section 4. We then proceed to analyze the security of a simplified version of SRP in Section 5. We revisit a known attack on earlier protocol versions in Section 6.1 in order to motivate the introduction of the aSDH assumption under which the SRP-6a protocol is secure. In Section 6.2, we outline the angel-based UC framework and subsequently provide a security proof of SRP-6a within this framework. We conclude with a discussion of our assumptions and their implications in Section 6.3.

2 The aPAKE Functionality

We use the UC framework [24] to analyse security of the SRP protocol. The UC model has become the de-facto standard when analysing the security of password-based protocols such as aPAKE, as it requires no assumptions on password distributions, and ensures security when composed with other protocols.

UC security builds upon the simulation paradigm by introducing the concept of an external *environment*. This environment monitors interactions between an adversary and multiple instances of a protocol Π. A protocol Π is considered to UC-realize a functionality $\mathcal{F}$, if for every efficient adversary $\mathcal{A}$ interacting with the real protocol Π there is a polynomial time simulator SIM interacting

only with the ideal functionality $\mathcal{F}$ such that the environment cannot distinguish both worlds. Here, the ideal functionality is an ideal version of the protocol which captures the protocol's desired functionality and is secure by design.

The first ideal functionality for aPAKE was introduced by Gentry et al. [36] and has since been adjusted various times, most recently by Hesse [41] and Gu et al. [37]. We use a slightly modified version of the ideal functionality $\mathcal{F}_{\mathsf{aPAKE}}$ from [37] which models explicit entity authentication and supports multiple clients. The functionality is shown in Fig. 1.

$\mathcal{F}_{\mathsf{aPAKE}}$ models the asymmetric password-authenticated key exchange between a client C and a server S. To register a user account, the server sends the user account identifier uid and password pw to the $\mathsf{StorePwdFile}$ interface of $\mathcal{F}_{\mathsf{aPAKE}}$ upon which the functionality creates an internal password file $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$. To initiate a key exchange, both parties access the functionality differently. While the client can input a fresh password for every new $\mathsf{CltSession}$ query, the server indicates the uid of the client in his $\mathsf{SvrSession}$ query which points to the password file $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$. In both cases, $\mathcal{F}_{\mathsf{aPAKE}}$ creates internal records $\langle \mathsf{ssid}, \mathsf{P}, \mathsf{P}', \mathsf{uid}, pw, \mathsf{role} \rangle$ where the parameter role is used for explicit authentication and is set to be 0 for the client and 1 for the server. In a key exchange session where there is no active attack, the NewKey interface of $\mathcal{F}_{\mathsf{aPAKE}}$ provides both parties with a uniformly random session key K if the client's password matches the password used by the server during registration, and a rejection symbol $\perp$ otherwise. The management of roles allow explicit entity authentication. The client only receives a key K if the same key K was also sent to the server, and $\perp$ otherwise. The functionality also models the following attack scenarios:

Online Guessing Attacks. The functionality $\mathcal{F}_{\mathsf{aPAKE}}$ models the inevitable online guessing attack through the $\mathsf{TestPwd}$ interface. The adversary can access this interface using his password guess for party P. If he guesses the party's password correctly, the functionality allows him to determine the session key output of P using the NewKey interface. Otherwise, the internal record stored by $\mathcal{F}_{\mathsf{aPAKE}}$ will be set to $\mathsf{interrupted}$ and the session key output of P will be $\perp$. Since $\mathcal{F}_{\mathsf{aPAKE}}$ always uses the server's registration password in his internal session record for a $\mathsf{SvrSession}$ query, the $\mathsf{TestPwd}$ interface also allows to model malicious servers which do not use the registration password. The functionality only allows one password guess against every honest party, modeling the fact that password guessing is an *online attack* in which the attacker engages in a protocol execution with an honest party.

Offline Dictionary Attacks. Another real-world threat to aPAKE protocols is server compromise, i.e., an adversary stealing the password file stored by the server, which can lead to an offline dictionary attack. $\mathcal{F}_{\mathsf{aPAKE}}$ models the theft of a password file with the $\mathsf{StealPwdFile}$ interface which changes the internal status of the password file to $\mathsf{compromised}$. In order to model offline dictionary attacks, the functionality offers an $\mathsf{OfflineTestPwd}$ interface. As opposed to the strong aPAKE functionality introduced in [47], $\mathcal{F}_{\mathsf{aPAKE}}$ allows for precomputation of dictionary attacks. This means the adversary can access the $\mathsf{OfflineTestPwd}$

interface using his password guess at any time, especially before compromising the server's password file. In case the password file is already compromised and the adversary correctly guesses the password via an $\mathsf{OfflineTestPwd}$ query, $\mathcal{F}_{\mathsf{aPAKE}}$ notifies him of his correct guess immediately. In case the $\mathsf{OfflineTestPwd}$ query happens before compromise, the guess will be logged by the functionality as $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw^* \rangle$. Then, in the event of a compromise, $\mathcal{F}_{\mathsf{aPAKE}}$ immediately informs the adversary of the correct password if he logged a correct guess before.

Server Impersonation Attacks. Additionally, an adversary in possession of the password file is able to establish a session key with the client on the server's behalf. This is called an impersonation attack. After successfully stealing the password file via a $\mathsf{StealPwdFile}$ query to $\mathcal{F}_{\mathsf{aPAKE}}$, the $\mathsf{Impersonate}$ query allows the adversary to compromise a session and determine the session key of the client via a NewKey query if the client uses the same password that was used to create the password file.

Explicit Entity Authentication. The $\mathcal{F}_{\mathsf{aPAKE}}$ functionality provides both parties either with a key K or with a rejection symbol. If the server receives a key $K \neq \perp$ then there is a corresponding client session with the same password. If the client receives a key $K \neq \perp$ then there is a corresponding server session with the same password and the server was sent the same key K. Considering that parties are authenticated through their knowledge of the shared password (file), receiving a key is equivalent to authenticating the counterparty's identity, thus modelling explicit entity authentication. This feature also allows the server to implement rate-limiting techniques in the protocol since he is informed whether a key exchange fails or not.

On the Server Input during Registration. All aPAKE functionalities introduced so far (e.g. [37,41,47]) abstract the user registration process away from the security analysis. This is modeled by letting the server input the user password to the functionality during registration, i.e., the server learns the password in clear during registration. This might look surprising as most aPAKE protocols – including OPAQUE and SRP – allow for a registration where the password remains hidden towards the server. While one could strengthen the functionality accordingly, we decided to follow the established convention and use the same model for registration as all other aPAKE works so far.

2.1 Changes to the Functionality

We make the following minor changes to the functionality from [37]. Our version of the aPAKE functionality is the first to support multiple users and models explicit authentication in a way that is compliant with the UC framework.

Multi-Client Setting. We adapt the aPAKE functionality to the multi-client setting, which we consider to be crucial for the targeted application. Initially, aPAKE functionalities [36,41] did not contain the notion of usernames in their

model, and instead assumed that each functionality identified through a session identifier sid is uniquely used by a single client. In the recent work by Gu et al. [37], this was changed to explicitly include a client-specific username uid as input to the interfaces and in the handling of the password files. However, this uid now fully replaced the standard sid, and [37] use sid to reference individual *sub-sessions* triggered by the user. That is, the functionality is again only usable by a single client. We revert to the more established handling of using sid as the globally unique ID that identifies the functionality, and use ssid to handle different sub-sessions, which makes the functionality in line with the UC framework with joint state [27]. This allows us to model a multi-client setting: one functionality with session id sid is tied to one server S and multiple clients C who each register at the server using an additional input user id uid. Consequently, when initiating a new session through the CltSession query, the client C needs to specify the corresponding uid for which the session is being started. For the sake of brevity, we make the sid not explicit in our functionality, but simply assume as a writing convention that sid is contained in every input to $\mathcal{F}_{\mathsf{aPAKE}}$ and used in all records.

Modelling Explicit Authentication. In aPAKE protocols with explicit authentication, typically one of the parties sends a failure message $\perp$ to the other party, in case of a failed authentication. A passive adversary observing the network traffic thus learns that the key exchange failed and will be able to deduce the session key $\perp$. Hence, the ideal functionality $\mathcal{F}$ in this case also needs to inform the adversary of a key exchange failure, in order to be realizable. Hesse [41] models this by having the functionality send $\perp$ to both the adversary and to the corresponding party in case of a failed session. However, this is technically not allowed in the UC framework, where a machine cannot invoke more than one other machine due to the requirement to be a single-threaded execution [24]. In the subsequent work by Jarecki et al. [37] this output was omitted from $\mathcal{F}$, but now misses the information that is needed to simulate failed sessions. Our solution uses the concept of *delayed outputs* as described in [24]. In our model, if a session fails, the functionality sends $(\mathsf{ssid}, \perp)$ to party $\mathcal{P}$ as a *public delayed output*. In a public delayed output the adversary decides when the output is sent to $\mathcal{P}$ and learns the output. Otherwise, the key (ssid, K) is sent as a *private delayed output* to $\mathcal{P}$. In a private delayed output, the adversary still decides when the output is sent to $\mathcal{P}$, but does not learn the output.

3 Preliminaries

Notation. With $\xleftarrow{r}$ we denote uniformly random sampling from a set. With $\mathbb{F}_p$ we denote a finite field of prime order p with $p > 2$ and with $\mathbb{F}_p^*$ we denote the multiplicative group $(\mathbb{F}_p \setminus \{0\}, \cdot)$. Throughout the paper we denote with g a primitive element of $\mathbb{F}_p$, i.e. a generator of the group $\mathbb{F}_p^*$. With respect to g, we define $\mathsf{cdh}_{\mathsf{g}}(g^a, g^b) := g^{ab}$ and $\mathsf{ddh}_{\mathsf{g}}(g^a, g^b, g^c) = 1$ iff $c = ab$. We denote with λ the security parameter.

Password Registration

– On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$ from S, create record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked fresh.

Stealing Password Data

– On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if there is no record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, return "no password file". Otherwise mark this record $\mathsf{compromised}$, and if there is a record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$ then send pw to $\mathcal{A}$.
– On $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw^*)$ from $\mathcal{A}$, do:
  • If $\exists$ record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$, do the following: If $pw^* = pw$ return "correct guess" to $\mathcal{A}$, else return "wrong guess".
  • Else, record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw^* \rangle$.

Password Authentication

– On $(\mathsf{CltSession}, \mathsf{ssid}, \mathsf{S}, \mathsf{uid}, pw')$ from C, if there is no record $\langle \mathsf{ssid}, \mathsf{C}, \ldots \rangle$ then record $\langle \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{uid}, pw', 0 \rangle$ marked fresh and send $(\mathsf{CltSession}, \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{A}$.
– On $(\mathsf{SvrSession}, \mathsf{ssid}, \mathsf{C}, \mathsf{uid})$ from S, if there is no record $\langle \mathsf{ssid}, \mathsf{S}, \ldots \rangle$ then retrieve record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, and if it exists then create record $\langle \mathsf{ssid}, \mathsf{S}, \mathsf{C}, \mathsf{uid}, pw, 1 \rangle$ marked fresh and send $(\mathsf{SvrSession}, \mathsf{ssid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ to $\mathcal{A}$.

Active Session Attacks

– On $(\mathsf{TestPwd}, \mathsf{ssid}, \mathsf{P}, pw^*)$ from $\mathcal{A}$, if there is a record $\langle \mathsf{ssid}, \mathsf{P}, \mathsf{P}', \mathsf{uid}, pw, \mathsf{role} \rangle$ marked fresh, then do: If $pw^* = pw$ then mark it $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess".
– On $(\mathsf{Impersonate}, \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if there is a record $\langle \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{uid}, pw, 0 \rangle$ marked fresh, then do: If there is a record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$ then mark $\langle \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{uid}, pw, 0 \rangle$ $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess".

Key Generation and Authentication

– On $(\mathsf{NewKey}, \mathsf{ssid}, \mathsf{P}, K^*)$ from $\mathcal{A}$, if there is a record $\mathsf{rec} = \langle \mathsf{ssid}, \mathsf{P}, \mathsf{P}', \mathsf{uid}, pw, \mathsf{role} \rangle$ not marked $\mathsf{completed}$, then do:
  • If rec is $\mathsf{compromised}$ set $K \leftarrow K^*$;
  • Else if $\mathsf{role} = 0$, rec is fresh, there is a record $\langle \mathsf{ssid}, \mathsf{P}', \mathsf{P}, \mathsf{uid}, pw, 1 \rangle$ s.t. $\mathcal{F}_{\mathsf{aPAKE}}$ sent (ssid, K') to P' while that record was marked fresh, set $K \leftarrow K'$;
  • Else if $\mathsf{role} = 1$, rec is fresh, there is a record $\langle \mathsf{ssid}, \mathsf{P}', \mathsf{P}, \mathsf{uid}, pw, 0 \rangle$ which is marked fresh, pick $K \xleftarrow{r} \{0,1\}^\lambda$;
  • Else set $K \leftarrow \perp$.

  Finally, mark rec as $\mathsf{completed}$. If $K = \perp$, provide public delayed output $(\mathsf{ssid}, \perp)$ to P, otherwise provide private delayed output (ssid, K) to P.

Fig. 1: The ideal Functionality $\mathcal{F}_{\mathsf{aPAKE}}$ (change to functionality from KHAPE [37] highlighted). As a writing convention we assume that the session id sid is contained in every input to $\mathcal{F}_{\mathsf{aPAKE}}$ and we do not include it explicitly in our description. The functionality is tied to a server S, and we assume that S is encoded in sid, e.g. as $\mathsf{sid} = (\mathsf{S}, \mathsf{sid}')$ for a unique sid'.

Cryptographic Assumptions. The security of SRP is based on the hardness of the well known CDH and Gap CDH assumptions, albeit defined over a finite field. A formal definition of these assumptions can be found in Appendix A. Here, we state them informally.

We consider hardness relative to an instance generator $\mathcal{G}$ which takes input 1^λ and outputs an instance $(\mathbb{F}_p, p, g)$, where $\mathbb{F}_p$ is a finite field of prime order p and g is a primitive element of $\mathbb{F}_p$.

CDH assumption Given $(\mathbb{F}_p, p, g) \leftarrow \mathcal{G}(1^\lambda)$ and tuple (g, g^a, g^b) for $a, b \xleftarrow{r} \mathbb{Z}_{p-1}$, it is computationally infeasible for every efficient adversary to compute the element g^{ab}.

Gap CDH assumption Given $(\mathbb{F}_p, p, g) \leftarrow \mathcal{G}(1^\lambda)$ and tuple (g, g^a, g^b) for $a, b \xleftarrow{r} \mathbb{Z}_{p-1}$, it is computationally infeasible for every efficient adversary to compute the element g^{ab}, even with access to a DDH oracle which on input (g^x, g^y, g^z) returns 1 if $g^z = g^{xy}$.

Note that these are the classical DH-related assumptions restricted to the multiplicative group $\mathbb{F}_p^*$ which has $p-1$ elements. In order for these assumptions to be hard, $p-1$ needs to have at least one large prime factor, otherwise it can be efficiently solved using the Pohlig Hellman algorithm [53]. This is achieved if $\mathcal{G}$ samples p as a safe prime, i.e. $p = 2q + 1$ for a prime number q.

In Appendix A, we also give a definition of the related DDH assumption. This assumption is known not to hold in finite fields due to an attack using the Legendre symbol [19,29]. In Section 6.3, we explore this attack and use it to assess the plausibility of our angel-based security framework, which relies on a DDH oracle in the finite field $\mathbb{F}_p$.

4 The SRP-6a Protocol

The SRP-6a protocol [71] is the most recent version of the SRP protocol family. It is an asymmetric PAKE protocol allowing two parties to compute a common key. SRP-6a uses a variation of the Diffie Hellman key exchange that includes a password-derived exponent. For our analysis, we use the protocol from [71] with optimized message-ordering. The protocol is depicted in Fig. 2 and we give a brief description of the protocol steps here.

The protocol works in a finite field of prime order p with primitive element g and consists of two phases. In an initialization or setup phase, the server registers a client with user id uid and password pw by storing the password file (s, v) where s is a random client-specific salt and v is the client's password verifier computed as $v := g^x$ where $x := H_1(s, \mathsf{uid}, pw)$ is a salted hash of the password.

In the login phase, both parties perform a Diffie Hellman key exchange with a twist. First, the server looks up the client's password file to retrieve the corresponding password verifier and salt. Then he samples a random element b from $\mathbb{Z}_{p-1}$, computes $B := kv + g^b$ and sends it together with the salt to the client. Here, k is a hash of public parameters. The client also generates a random number $a \xleftarrow{r} \mathbb{Z}_{p-1}$, sets $A := g^a$ and uses the salt he received from the server and his

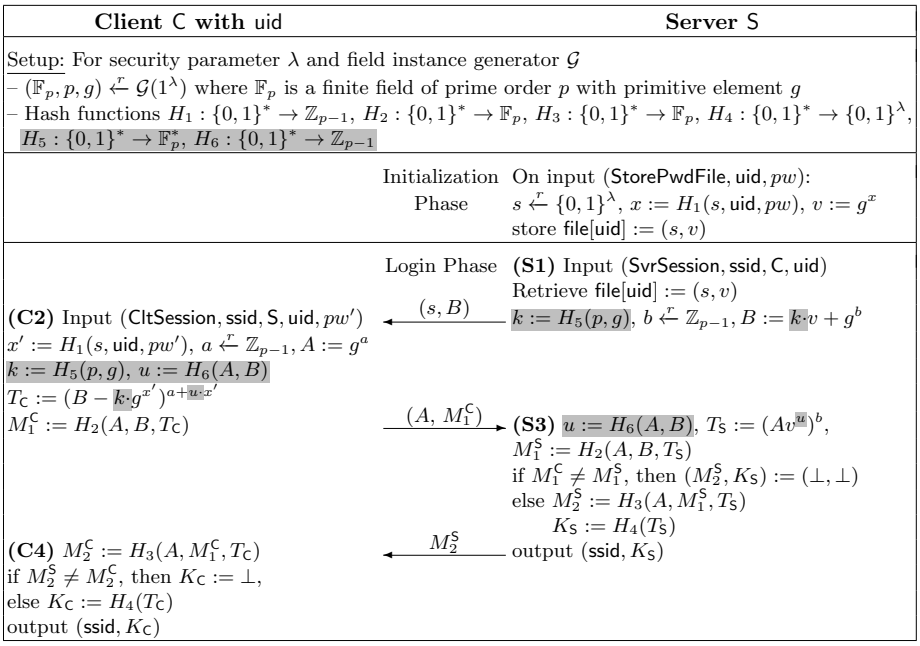

Fig. 2: The SRP-6a Protocol with UC notation and optimized message ordering as described in [71]. To avoid confusion with the server S we changed the notation of the ephemeral session key to be T_C for the client and T_S for the server. We assume all messages and inputs to the hash functions are prefixed by the sid, and all messages and hash function inputs specific to the login phase are further prefixed by ssid. We omit these values in the description as a writing convention. Removing the shadowed text yields the SRP_{base} protocol.

password to compute x'. He is now able to compute an ephemeral session key $T_C := (B - kg^{x'})^{a+ux'}$, where u is a hash of protocol messages A and B. The client then sends his ephemeral public key A together with a key confirmation message M_1^C to the server. The server can compute the same ephemeral session key in a different way by calculating $T_S := (Av^u)^b$ and sends key confirmation message M_2^S to the client. Both parties derive a common session key by hashing the ephemeral session key if the check of the counterparty's key confirmation message is successful. Correctness of the protocol follows from the fact that in case of matching passwords ($pw = pw'$) it holds that $v = g^{x'}$, thus

$$T_C = (B - kg^{x'})^{a+ux'} = (kv + g^b - kg^{x'})^{a+ux'} = g^{ab+bux'}$$
$$T_S = (Av^u)^b = (Ag^{x'u})^b = (g^{a+ux'})^b = g^{ab+bux'}$$

and both parties derive the same session key $K_C = H_4(T_C) = H_4(T_S) = K_S$. If the password of the client does not match the server's password it holds that $T_S \neq T_C$ and both parties output a rejection symbol $\perp$.

On the Necessity of a Finite Field. In the description of the SRP protocol, there is some confusion whether the SRP protocol works in a group, ring or finite field. In [73], the protocol SRP-3 is described in the context of a finite field while in version SRP-6 [71], it is claimed that the protocol works in a group. In the line of SRP literature this confusion persists. The SRP protocol is claimed to either work in a group [31,40,61], ring [49,69,70] or finite field [60,72]. SRP is always instantiated in the finite field $\mathbb{Z}_p$ for p prime, so this confusion merely exists in the description. We stress that SRP indeed needs to be instantiated in a finite field because it (a) uses both addition and multiplication operations and (b) requires a generator of the multiplicative subgroup of all non-zero field elements in order to sample A and B.

Resistance to Non-targeted Offline Dictionary Attacks. The SRP protocol uses a client-specific random salt in the creation of the password verifier. This offers some resistance to precomputation offline dictionary attacks. However, since the salt is sent in the clear during login, the SRP protocol is still vulnerable to targeted precomputation offline dictionary attacks where an adversary is able to build client-specific databases using the salt.

On the Server-Side Creation of the Password Verifier. In the formal analysis of aPAKE protocols the model usually assumes that the server receives a client's password during initialization and uses it to register the client. In practical applications however, it is common that the client performs the initialization computations on his device and sends only the password file to the server (as done in the 1Password implementation [35]). This way, the server never even sees the password which is clearly beneficial. In this case, the first message must be sent over a secure channel since it contains the password file and could be used by an eavesdropping adversary to mount offline dictionary attacks.

UC-fying the protocol. Since the UC model requires unique (sub)session identifiers sid and ssid as input to the protocol, our analysis covers a slightly modified version of the SRP protocol that incorporates these values. The purpose of the session identifier sid is to uniquely identify separate protocol instances which is needed for the composability guarantees in UC. The subsession identifier ssid serves to distinguish separate login sessions and allows to recognize messages belonging to the same multi-round protocol. As the UC functionality expresses its security guarantees with respect to these subsessions, the ssid must also become part of the cryptographic protocol and session-specific values need to be bound to this identifier. We do this by simply including the ssid (and sid) as input to hash function, which is the standard approach for handling these identifiers. Further details on the practical implications of incorporating these UC artifacts, as well as the potential security consequences of their omission, are discussed in Section 6.3.

5 Warmup: Security of Simplified SRP

In this section, as a warm-up, we analyse the security of SRP against *semi-passive* adversaries which are allowed to eavesdrop on any session and steal the password files held by the server. First, we argue that the security proof used in [73] to claim security of SRP-3 against eavesdropping attacks is flawed. We show how this proof can be fixed and augmented to show UC security of a simplified version of SRP called SRP_{base}. This is the most simple version of SRP that we can show secure against semi-passive adversaries under standard assumptions. We argue that security of SRP_{base} implies security of the SRP versions 3, 6 and 6a. Formally, we show SRP_{base} secure in the UC framework using a modified ideal functionality $\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$.

Flawed Reduction in SRP-3. In [73], the author establishes the security of SRP against passive eavesdropping attacks by a reduction to the CDH problem. However, the reduction uses hard-coded values as input to the eavesdropping adversary. Therefore it rules out only adversaries who break the protocol on *these* hard-coded values. In the proof of Theorem 1, we use a variation of their reduction where we randomize the input to the adversary. We describe the flawed reduction in more detail in Appendix B.

The SRP_{base} Protocol. We introduce SRP_{base}, a simplified version of SRP-6a where the scrambling parameters u and k are removed. In SRP_{base}, the intermediary session key is computed as $T_{\mathsf{C}} = (B - g^{x'})^{a+x'}$ by the client and as $T_{\mathsf{S}} = (Av)^b$ by the server. The protocol is depicted in Fig. 2.

Passive Security of SRP_{base}. We show SRP_{base} secure against semi-passive attacks. The adversary in our model serves as a mere network attacker who observes all communication between honest parties C and S. On top of that, the adversary is allowed to compromise the server's password file and may learn the password associated to it by offline brute force attacks. This strengthens the ability of the adversary compared to a passive adversary and models the real world threat of an adversary obtaining stolen password files through darknet databases. The adversary is still passive in the sense that he is not allowed to actively attack any session and engage in a key exchange. In order to capture this attack model, we introduce a modified functionality $\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$ which is identical to $\mathcal{F}_{\mathsf{aPAKE}}$ but with the active session attack interfaces removed.

To show that a protocol securely UC-realizes this ideal functionality, one needs to show essentially two things:

1. There is a simulator who can produce a transcript of a protocol run between two honest parties that is indistinguishable from a real transcript without knowing the passwords of the parties. The simulator only receives the information whether the key exchange failed.
2. The simulator can detect offline password tests by the adversary. Furthermore, he can (a) generate an indistinguishable password file if he is informed

about the correct password upon server compromise or (b) generate an indistinguishable non-committing password file upon server compromise which he can later adapt to match the correct password of the client.

Theorem 1. *Let $\mathcal{G}$ be a finite field instance generator. If the CDH problem is hard relative to $\mathcal{G}$ and if the real-world adversary is restricted to semi-passive attacks, then SRP_{base} instantiated with $\mathcal{G}$ securely UC-realizes the functionality $\mathcal{F}_{\mathsf{aPAKE}}^{passive}$ in the random oracle model. In detail, it holds that*

$$\Big|\Pr[Real_{\mathcal{Z}}(SRP_{base}, \mathcal{A})] - \Pr[Ideal_{\mathcal{Z}}(\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}, \mathsf{SIM})]\Big| \leq$$
$$\frac{l_{H_1}^2}{p} + \frac{l_{H_2}^2}{p} + \frac{l_{H_3}^2}{p} + \frac{l_{H_4}^2}{2^\lambda} + (l_{H_1} + 2l_{H_2} + l_{H_3}) \cdot \boldsymbol{Adv}_{\mathcal{B}_{\mathsf{CDH}}}^{\mathsf{CDH}}(\lambda) + \frac{l_{\mathsf{ssid}}}{p}$$

where l_{H_i} corresponds to the number of hash queries to H_i for $i = 1, ...4$, p is the order of the finite field, λ is the security parameter and l_{ssid} is the maximum number of subsessions between S and uid.

Proof Sketch. The proof follows a sequence of games starting with the real execution of the protocol where a simulator handles the execution using the parties' private inputs, i.e. their passwords. This execution is subsequently changed and ends with an ideal execution of the protocol where the simulator only interacts with the ideal functionality $\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$. We show all of these changes to be indistinguishable by the environment, from which the UC-security of $\mathrm{SRP}_{\mathrm{base}}$ follows.

The main idea of the simulation is to replace protocol values B, M_1^{C} and M_2^{S} with random values sampled from the same domain as the real values, thus making them independent of the parties' passwords. To simulate the last flow of the protocol, the simulator uses the information, whether the functionality provides a public or a private delayed output after a NewKey query. If the output is public, the passwords of the parties differ and the simulator sets M_2^{S} to be the rejection symbol $\perp$. Otherwise M_2^{S} is chosen as a randomly sampled field element. We briefly highlight the indistinguishability arguments for the replaced values:

Indistinguishability of $M_1^{\mathsf{C}}, M_2^{\mathsf{S}}, K_{\mathsf{C}}, K_{\mathsf{S}}$: Assuming that the CDH assumption holds relative to $\mathcal{G}$, we argue that no adversary can distinguish whether these values are computed as in the real world or sampled uniformly at random. In the real world, it holds that $M_1^{\mathsf{C}} = H_2(A, B, T_{\mathsf{C}})$, $M_2^{\mathsf{S}} = H_3(A, M_1^{\mathsf{S}}, T_{\mathsf{S}})$, $K_{\mathsf{C}} = H_4(T_{\mathsf{C}})$ and $K_{\mathsf{S}} = H_4(T_{\mathsf{S}})$. Note that the simulator only replaces values M_1^{C} and M_2^{S} with random values while K_{C} and K_{S} are provided as random values by $\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$ upon a NewKey query. Since M_2^{S} is only sent in the protocol if the passwords of both parties match, it holds in this case that $M_1^{\mathsf{S}} = M_1^{\mathsf{C}}$ and $T_{\mathsf{S}} = T_{\mathsf{C}}$. The same holds for K_{C} and K_{S} which are only distinct from $\perp$ if the passwords of both parties match. Replacing all of these values with random values is then indistinguishable in the random oracle model unless the adversary is able to query H_2 on input (A, B, T_{C}), H_3 on input $(A, M_1^{\mathsf{C}}, T_{\mathsf{C}})$ or H_4 on T_{C}. In each of these cases, the adversary has to compute $T_{\mathsf{C}} := \mathsf{cdh}_{\mathsf{g}}(Ag^{x'}, B - g^{x'})$ for $x' := H_1(s, \mathsf{uid}, pw')$ which is hard if the CDH assumption holds relative to $\mathcal{G}$.

The idea of the reduction is as follows: Let $\mathcal{B}_{\mathsf{CDH}}$ be an adversary trying to break a CDH challenge. $\mathcal{B}_{\mathsf{CDH}}$ plays the role of the simulator. He simulates both the client and the server and controls the output of the hash oracle, which allows him to embed a CDH challenge (g, g^d, g^e) into the protocol. He sets $A := g^d$ and $B := g^e + g^{x'}$ for the hash value $x' := H_1(s, \mathsf{uid}, pw')$. For the reduction, we assume that $\mathcal{B}_{\mathsf{CDH}}$ knows the password pw' of the client. Recall that at this point of the proof, the simulator (and thus $\mathcal{B}_{\mathsf{CDH}}$) still knows the parties' private inputs. This is subsequently changed until the simulator is able to simulate the whole protocol execution without knowing the private inputs. In order to distinguish whether $M_1^{\mathsf{C}}, M_2^{\mathsf{S}}, K_{\mathsf{C}}, K_{\mathsf{S}}$ are as in the real execution of the protocol or chosen at random, there must be an adversary $\mathcal{A}$ who queries $T_{\mathsf{C}} := \mathsf{cdh}_{\mathsf{g}}(Ag^{x'}, B - g^{x'}) = \mathsf{cdh}_{\mathsf{g}}(g^{d+x'}, g^e) = g^{de+ex'}$ to one of the random oracles H_2, H_3 or H_4. But then $\mathcal{B}_{\mathsf{CDH}}$ can give a solution to the CDH challenge with non-negligible success probability. To do so, $\mathcal{B}_{\mathsf{CDH}}$ chooses one of the random oracle inputs from $\mathcal{A}$ at random and outputs $T_{\mathsf{C}} \cdot ((g^e)^{x'})^{-1}$ for the corresponding T_{C} value.

Indistinguishability of B: The real world value of B leaks some information about the password verifier v since B is set as $B := v + g^b$ for a random $b \xleftarrow{r} \mathbb{Z}_{p-1}$. This equivalents to sampling uniformly at random from $\mathbb{F}_p \setminus \{v\}$ and means that B can never be equal to v. In the ideal world, on the other hand, the simulator samples B uniformly at random from $\mathbb{F}_p$. We argue that the information leaked is negligible and does not help an adversary in distinguishing the two worlds. Furthermore, in the ideal world, B is not further used in the protocol since M_1^{C} and M_2^{S} which were dependent on B in the real world, are also sampled from random in the ideal world. Thus, both worlds are only distinguishable if the simulator picks B such that $B = v$ holds. This happens only with negligible probability, since v is instantiated as $v := g^x$ for a uniformly random H_1 output x from $\mathbb{Z}_{p-1}$.

Server compromise: We observe that the simulator is able to detect offline password tests from queries to the hash function H_1. An offline password test for a client with user account identifier uid corresponds to a query $H_1(s, \mathsf{uid}, pw)$ using the client-specific salt s. Upon the first call to either $\mathsf{SvrSession}$ or $\mathsf{StealPwdFile}$, the simulator chooses a random salt and creates a password file $\mathsf{file}[\mathsf{uid}] = (s, \perp)$. This allows the simulator to detect offline password tests in queries to H_1 and pass them to the ideal functionality using a $\mathsf{OfflineTestPwd}$ query. In case of a successful offline password guess there are two cases:

1. The adversary has queried the correct password pw before server compromise. In this case, the functionality informs the simulator about the correct password in the moment of compromise. This allows the simulator to retrieve the output of the corresponding H_1 query $(H_1, [s, \mathsf{uid}, pw], h)$ and set the password verifier to $v := g^h$ which is identical to the real world.
2. The adversary queries the correct password pw after server compromise. In this case, in the moment of server compromise, the simulator chooses a random x and sets the password verifier to $v := g^x$. When the ideal functionality then informs the simulator about a correct guess upon a successful $\mathsf{OfflineTestPwd}$ query, the simulator uses his ability to program the random

oracle and sets the output of this query to x. Again this yields a view to the environment that is identical to the real world execution of the protocol.

Apart from the reprogramming operation in H_1, the random oracles are simulated by lazy sampling of output queries and aborting on a collision. This explains the first four terms of the bound in the theorem. The fifth term corresponds to the adversary distinguishing the game by computing T_{C} which would break the CDH assumption in $\mathbb{F}_p$. The last term of the bound corresponds to the adversary's ability to distinguish the games if $B = v$ for any subsession between client and server.

In Fig. 3, we describe common interfaces which are used by the simulator in this theorem as well as by the simulator for Theorem 2 in Section 6. The simulation of the login phase is given in Fig. 4, and the full proof is given in Appendix E.

6 Security Analysis of SRP-6a against Active Adversaries

In this section, we prove security of the SRP-6a protocol against active adversaries in the angel-based UC framework. As a stepping stone, we first show why security of $\mathrm{SRP}_{\mathrm{base}}$ can only be shown in the presence of a semi-passive adversary. There are two known attacks involving active adversaries that also apply to $\mathrm{SRP}_{\mathrm{base}}$. In [73], Wu describes a client impersonation attack on SRP-1 where an adversary uses a stolen verifier to authenticate as a client and in [71], Wu describes a two-for-one-guessing attack where a malicious server is able to online guess two passwords in one protocol run. These attacks led to the introduction of scrambling parameter u and k. In order to show security of the SRP-6a protocol, we introduce a new assumption which eliminates two-for-one-guessing attacks. We motivate this assumption by highlighting the two-for-one-guessing attack, while the details of the client impersonation attack can be found in Appendix C.

6.1 Introduction of k: Prevent Two-for-one-Guessing Attack

A corrupt server in an aPAKE protocol can always test one password per protocol run against the client's login password. In SRP-3, the server tests a password pw^* by computing $x^* := H_1(s, \mathsf{uid}, pw^*)$, $v^* := g^{x^*}$ and running the protocol with test password file (s, v^*). If for the client's message M_1^{C} it holds that $H_2(A, B, T_{\mathsf{S}}) = M_1^{\mathsf{C}}$ for $T_{\mathsf{S}} = (A(v^*)^u)^b$, the server has guessed the password correctly.

Due to the symmetry in the creation of B however, in SRP-1 and SRP-3, a malicious server may test two password guesses in one protocol run. To understand the attack, first observe that, given $B = g^x + g^b$ the server can compute both $T_{\mathsf{S},1} = (A(g^x)^u)^b$ and $T_{\mathsf{S},2} = (A(g^b)^u)^x$. This allows him to test two passwords pw_1, pw_2 by setting $B := g^{x_1} + g^{x_2}$ for $x_1 = H_1(s, \mathsf{uid}, pw_1)$ and $x_2 = H_1(s, \mathsf{uid}, pw_2)$. Then, the server can compute the values $T_{\mathsf{S},1}$ and $T_{\mathsf{S},2}$ and check if either of them satisfies $M_1^{\mathsf{C}} = H(A, B, T_{\mathsf{S},i})$. If the condition holds for one of the values, the server has guessed the password correctly.

Password Registration

– *Convention:* First call to $\mathsf{SvrSession}$ or $\mathsf{StealPwdFile}$ for $(\mathsf{S}, \mathsf{uid})$: pick $s \xleftarrow{r} \{0,1\}^\lambda$ and record $\mathsf{file}[\mathsf{uid}] := (s, \perp)$.

Stealing Password File

– On $\mathcal{Z}$'s permission to do so, send $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}$. If $\mathcal{F}$ sends "no password file", SIM passes it to $\mathcal{A}$, otherwise declares $(\mathsf{S}, \mathsf{uid})$ $\mathsf{compromised}$ and:
  1. If $\mathcal{F}$ returns pw, retrieve record $(H_1, [s, \mathsf{uid}, pw], h)$ and set $x := h$; else, pick $x \xleftarrow{r} \mathbb{Z}_{p-1}$
  2. Set $v := g^x$
  3. If $k \cdot v = B_{\mathsf{ssid}}$ for some ssid aborts
  4. Update $\mathsf{file}[\mathsf{uid}] := (s, v)$ and send $\mathsf{file}[\mathsf{uid}]$ to $\mathcal{A}$.

Simulation of Random Oracles

On $H_1(s, \mathsf{uid}, pw)$ from $\mathcal{A}$:
  if this is the first such query, then
    if $\exists$ $\mathsf{file}[\mathsf{uid}] = (s, *)$ send $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw)$ to $\mathcal{F}$:
      upon answer "correct guess", set $h := x$
      upon answer "wrong guess", set $h \xleftarrow{r} \mathbb{Z}_{p-1} \setminus \{x\}$
      else set $h \xleftarrow{r} \mathbb{Z}_{p-1}$
    else set $h \xleftarrow{r} \mathbb{Z}_{p-1}$
    if $\exists$ record $(H_1, [*, *, *], h)$ abort
    store $(H_1, [s, \mathsf{uid}, pw], h)$ and reply with h
  else retrieve record $(H_1, [s, \mathsf{uid}, pw], h)$ and reply with h

On $H_i(m)$ from $\mathcal{A}$ for $i \geq 2$:
  if this is the first such query, then
    pick $h \xleftarrow{r} \mathcal{D}(H_i)$
    if $\exists$ record $(H_i, [*], h)$ abort
    store $(H_i, [m], h)$ and reply with h
  else retrieve record $(H_i, [m], h)$ and reply with h

Fig. 3: Common simulation interfaces for the simulators of Theorem 1 and Theorem 2. B_{ssid} denotes the values of B used in subsession ssid. The scrambling parameter k is only used by the simulator of Theorem 2. The value x in H_1 denotes the value SIM chooses at random when the password file $v := g^x$ gets compromised, yet no previous H_1 query (s, uid, pw) for the correct password was made. $\mathcal{D}(H_i)$ denotes the output domain of H_i. Unless specified otherwise, the random oracles are simulated as above.

To mitigate this attack, SRP-6a [71] introduces the value k to remove the symmetry in the computation of B which is now computed as $B := kv + g^b$. The above attack would still work if the adversary knows the discrete logarithm of k, as he can set $B = kg^{x_1} + kg^{x_2}$ and use his knowledge of the discrete logarithm of kg^{x_2} and kg^{x_1} to test the two passwords. Thus, the original SRP-6 protocol proposed hardcoding $k = 3$, but in SRP-6a, k was changed to be the output of a hash function of public parameters such as p and g.

Login Phase

(S1) On $(\mathsf{SvrSession}, \mathsf{ssid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ from $\mathcal{F}$:
(G_1) retrieve $\mathsf{file}[\mathsf{uid}] = (s, v)$. If $\nexists\ \mathsf{file}[\mathsf{uid}]$ do nothing and halt
(G_4) pick $B \xleftarrow{r} \mathbb{F}_p$
(G_4) if $(\mathsf{S}, \mathsf{uid})$ is marked compromised and $B = v$, abort
(G_1) send (s, B) to $\mathcal{A}$ intended for C

(C2) On $(\mathsf{CltSession}, \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{F}$ and receiving (s, B) from $\mathcal{A}$:
(G_3) pick $A \xleftarrow{r} \mathbb{F}_p^*$, $M_1^{\mathsf{C}} \xleftarrow{r} \mathbb{F}_p$
(G_1) send (A, M_1^{C}) to $\mathcal{A}$ intended for S

(S3) Upon receiving (A, M_1^{C}) from $\mathcal{A}$:
(G_6) send $(\mathsf{NewKey}, \mathsf{ssid}, \mathsf{S}, \top)$ to $\mathcal{F}$:
(G_6) upon receiving public delayed output $(\mathsf{ssid}, \bot)$, set $M_2^{\mathsf{S}} := \bot$
(G_3) else set $M_2^{\mathsf{S}} \xleftarrow{r} \mathbb{F}_p$
(G_1) send M_2^{S} to $\mathcal{A}$ intended for C

(C4) Upon receiving M_2^{S}) from $\mathcal{A}$:
(G_6) send $(\mathsf{NewKey}, \mathsf{ssid}, C, \top)$ to $\mathcal{F}$

Fig. 4: Simulator SIM showing that $\mathrm{SRP}_{\mathrm{base}}$ realizes $\mathcal{F} = \mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$ with game numbers to indicate which game introduces a particular line of code. Fig. 3 describes the simulation of password registration, server compromise and random oracles. We assume the simulator forwards all private delayed outputs to the corresponding parties.

The SRP-Two-For-One-Guessing-Problem. We capture the hardness of mounting a two-for-one guessing attack in SRP-6a as the "SRP-two-for-one-guessing-problem". Therefore, we assume the adversary plays the role of the server and engages in a session with protocol values A, k, u. Now, in order to test two passwords pw_1, pw_2 with salted hashes x_1, x_2, the adversary has to find a value B that allows him to compute the two ephemeral session keys $T_1 = \mathsf{cdh}_{\mathsf{g}}(Ag^{x_1 u}, B - kg^{x_1})$ and $T_2 = \mathsf{cdh}_{\mathsf{g}}(Ag^{x_2 u}, B - kg^{x_2})$, i.e. $T_1 = g^{ab_1 + b_1 x_1 u}$ and $T_2 = g^{ab_2 + b_2 x_2 u}$ for $g^{b_1} := B - kg^{x_1}$, $g^{b_2} := B - kg^{x_2}$ and $g^a := A$. Since the adversary knows x_1, x_2 and u, we remove the terms $g^{x_1 u}$ and $g^{x_2 u}$ in the formalization of the problem which we state as follows:

Definition 1 (SRP-Two-For-One-Guessing-Problem). *We say the SRP-two-for-one-guessing-problem is hard relative to $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$*

$$\mathbf{Adv}_{\mathcal{A}}^{\textit{SRP-2-1}}(\lambda) := \Pr[\mathcal{A}(\mathbb{F}_p, p, g, A, k, x_1, x_2) = (B, T_1, T_2) : \mathsf{ddh}_{\mathsf{g}}(A, B - kg^{x_1}, T_1) = \mathsf{ddh}_{\mathsf{g}}(A, B - kg^{x_2}, T_2) = 1]$$

is negligible in λ, where $(\mathbb{F}_p, p, g) \xleftarrow{r} \mathcal{G}(1^\lambda), A \xleftarrow{r} \mathbb{F}_p^*, k \xleftarrow{r} \mathbb{F}_p^*$, $x_1 \xleftarrow{r} \mathbb{Z}_{p-1}$ and $x_2 \xleftarrow{r} \mathbb{Z}_{p-1}$.

Note that any adversary solving a "SRP-n-for-1-guessing problem" can be reduced to an adversary solving a two-for-one-guessing problem, thus the hardness of the two-for-one-guessing problem rules out the existence of any adversary trying to test multiple passwords in one protocol run.

The SRP-two-for-one-guessing-problem essentially requires the values in the second component of the DDH check to differ by a fixed value independent of B since $(B - kg^{x_1}) - (B - kg^{x_2}) = kg^{x_2} - kg^{x_1}$. This motivates the formalization of the equivalent problem of computing two CDH tuples where the difference of the second component of both instances equals a randomly chosen value. We call this the additive simultaneous CDH problem (aSDH):

Definition 2 (Additive Simultaneous CDH Problem (aSDH)). *We say that the aSDH problem is hard relative to $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$*

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{aSDH}(\lambda) := \Pr[\mathcal{A}(\mathbb{F}_p, p, g, X, \Delta) = (Y, Z_1, Z_2) : \\ \mathsf{ddh}_{\mathsf{g}}(X, Y, Z_1) = \mathsf{ddh}_{\mathsf{g}}(X, Y + \Delta, Z_2) = 1]\end{aligned}$$

is negligible in λ, with $(\mathbb{F}_p, p, g) \xleftarrow{r} \mathcal{G}(1^\lambda), X \xleftarrow{r} \mathbb{F}_p^*, \Delta \xleftarrow{r} \mathbb{F}_p^*$.

We formulate two natural extensions of the assumptions, the *Strong aSDH Assumption*, where the adversary additionally has access to a restricted DDH oracle with fixed input X (i.e. $\mathsf{ddh}_{\mathsf{g}}(X, \cdot, \cdot)$), and the *Gap aSDH Assumption* where the adversary has access to a full DDH oracle $\mathsf{ddh}_{\mathsf{g}}(\cdot, \cdot, \cdot)$. Since the adversary in the angel-based framework will have access to a full DDH oracle, our proof in Theorem 2 requires the gap version of the assumption. In Section 6.3, we argue that with some restrictions on the adversary, security can be shown under the strong aSDH assumption.

Relation to Known Assumptions. We conject that there is no efficient algorithm for solving the aSDH problem in finite fields where the CDH problem is hard. In order to solve this problem, the CDH tuple of X with two values Y and $Y + \Delta$ for a random value Δ needs to be computed. While either Y or $Y + \Delta$ can be chosen such that the discrete logarithm is known for one of these value, it is intuitively hard to determine the discrete logarithm (or a CDH tuple) for both of these values. Due to the attacker's ability to determine Y, we cannot use an aSDH attacker to solve a CDH problem and it seems there is no other known problem the aSDH problem can be reduced to since most CDH type assumptions work only in a group and not in a finite field. aSDH is the first such assumption that incorporates both field operations into its assumption. In Appendix D , we elaborate on why existing proof techniques to show hardness in generic models such as [8,20,45,46,62] cannot be applied to the aSDH assumption.

On the Necessity of the aSDH Assumption. When comparing assumptions across mathematical structures, aSDH is very similar to the assumption made in the proof of the PAKE protocols SPAKE1 and SPAKE2 [7] which also rules out multiple password guesses. Their security relies on a derivative of the chosen-basis CDH (CCDH) assumption. Here, an adversary is given group elements X, M, N and has to provide values (Y, Z_1, Z_2) such that $Z_1 = \mathsf{cdh}_{\mathsf{g}}(X, Y)$ and $Z_2 = \mathsf{cdh}_{\mathsf{g}}(X/M, Y/N)$. This is similar to the aSDH assumption as in both assumptions the adversary has to provide two valid CDH tuples which are connected by a bijective mapping.

As an alternative approach, it might appear tempting to simply weaken the ideal functionality to permit two password guesses in the hope of avoiding the new aSDH assumption. However, we would run into a similar problem in the security proof where we would have to find a way to provably limit the server to no more than two password guesses, which would require a similar (or even more tailored) assumption to the one we introduced.

6.2 UC Security of SRP-6a

In this subsection, we begin by presenting the angel-based UC framework and subsequently provide a proof that SRP-6a securely realizes the ideal functionality $\mathcal{F}_{\mathsf{aPAKE}}$ from Fig. 1, when a DDH-oracle (in finite fields) is available as angel.

Angel-Based UC Security. The UC framework provides strong security guarantees, but its stringent restrictions also limit what can be proven secure. In fact, for several functionalities, UC-security is provably not attainable [25,48]. To address this limitation, several relaxations of UC-security have been introduced. *Superpolynomial-time simulation (SPS)* [52] is a relaxation which allows the simulator to run in superpolynomial time. In *angel-based UC security* [54] and its generalization *UC security with a superpolynomial-time helper* [26], both the simulator and the adversary run in polynomial time, but have access to a "helper" oracle (the "angel") which can perform some pre-determined superpolynomial-time task such as inverting a one-way function. The latter two frameworks even preserve the composability features of the UC framework, albeit with the restriction that composability only holds among protocols that are secure with respect to the same helper oracle.

More precisely, angel-based UC security is identical to standard UC security, except that all parties have access to a global helper oracle or angel Φ with superpolynomial resources. A protocol Π is said to Φ-UC-realize functionality $\mathcal{F}$, if the environment cannot efficiently distinguish between an adversary interacting with the real protocol Π and a simulator interacting with the ideal functionality $\mathcal{F}$ where both the adversary and the simulator have access to helper oracle Φ.

In our proof of security, we utilize a DDH oracle as global helper oracle. We emphasize that the DDH oracle used in our proof is limited to the finite field employed by the SRP protocol. Thus, this DDH oracle cannot be used to break DDH assumptions in other groups where the DDH assumption is believed to hold. We discuss the necessity and impact of this additional DDH helper oracle in more detail in Section 6.3.

Extracting Passwords with the DDH Oracle. We use the DDH oracle in our security proof in order to extract passwords from adversarial messages. This was not needed in our proof for Theorem 1 since we considered only semi-passive adversaries who were not allowed to actively engage in a protocol session. However, in our security proof for SRP-6a, we extend our analysis to include active attacks where the adversary can interact with the simulator and inject protocol messages. This introduces a significant challenge commonly encountered in

proofs within the UC framework: the simulator must extract the adversary's private input used to generate these messages. Once the adversary's private input is extracted, the simulator can compare it with the honest party's input using the ideal functionality, and adjust the simulation accordingly.

In the proof of the SRP-6a protocol, the simulator requires a DDH oracle when simulating an honest server[2]. If an adversarial client sends protocol message (A, M_1^{C}), the simulator needs to extract the adversary's password from this message. To achieve this, the simulator first uses his ability to control the random oracle and extracts T_{C} from $M_1^{\mathsf{C}} = H_2(A, B, T_{\mathsf{C}})$. He then verifies whether T_{C} corresponds to a valid DH value $\mathsf{cdh}_{\mathsf{g}}(Ag^{ux}, B - kg^x)$ for any of the values $x = H_1(s, \mathsf{uid}, pw')$ which the adversary previously obtained from the random oracle. If a match is found, the simulator can extract the adversary's password pw' and verify it against the server's password using the TestPwd interface of the ideal functionality. If the passwords match, the simulator proceeds to simulate a successful key exchange, otherwise he simulates key exchange failure.

Proof of Security. We prove SRP-6a secure against active attacks in the angel-based UC security framework which utilizes a global DDH oracle $\Phi_{\mathcal{G}}$ (limited to the finite field output by $\mathcal{G}$). We further make the following assumptions on the adversary's capabilities:

- The adversary may actively engage in any session. He is allowed to alter or drop messages sent by honest parties and inject his own messages. That is, we do not assume a secure communication channel between client and server.
- Following the approach of previous aPAKE papers [37,47] we assume *static corruption* of the parties, that is, parties cannot change their corruption status throughout the protocol. However, $\mathcal{F}_{\mathsf{aPAKE}}$ allows the adversary to adaptively compromise a server via query StealPwdFile during protocol execution. This creates two separate notions of server corruption. If an *honest server* has his password file stolen, we call the corresponding aPAKE instance *compromised*, whereas we call a server *corrupt* if he is controlled by the adversary through static corruption.

Theorem 2. *Let $\mathcal{G}$ be a finite field instance generator and let $\Phi_{\mathcal{G}}$ be a DDH oracle for $\mathcal{G}$. If the Gap CDH problem and the Gap aSDH problem are hard relative to $\mathcal{G}$, then SRP-6a instantiated with $\mathcal{G}$ securely $\Phi_{\mathcal{G}}$-UC-realizes the functionality $\mathcal{F}_{\mathsf{aPAKE}}$ in the random oracle model w.r.t. static corruption. More precisely, for every adversary $\mathcal{A}$, there exist adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ against the Gap CDH and*

[2] To simplify our proof, the simulator therein also uses the DDH oracle to extract passwords when simulating an honest client. However, we point out in our discussion "Security without Angel" in Section 6.3 that this case can be handled without access to the DDH oracle.

Gap aSDH problems such that

$$\Big|\Pr[Real_{\mathcal{Z}}(SRP\text{-}6a, \mathcal{A}^{\Phi_{\mathcal{G}}})] - \Pr[Ideal_{\mathcal{Z}}(\mathcal{F}_{\mathsf{aPAKE}}, \mathsf{SIM}^{\Phi_{\mathcal{G}}})]\Big| \leq$$
$$\frac{l_{H_1}^2}{p} + \frac{l_{H_2}^2}{p} + \frac{l_{H_3}^2}{p} + \frac{l_{H_4}^2}{2^\lambda} + \frac{l_{H_5}^2}{p} + \frac{l_{H_6}^2}{p} + \frac{l_{\mathsf{ssid}}}{p} + \boldsymbol{Adv}^{\mathsf{GapCDH}}_{\mathcal{B}_1^{\Phi_{\mathcal{G}}}}(\lambda)$$
$$+ \frac{l_{H_1}^2}{2^{p-1}} + l_{H_2}^2 \cdot \boldsymbol{Adv}^{\mathsf{GapCDH}}_{\mathcal{B}_2^{\Phi_{\mathcal{G}}}}(\lambda) + l_{H_1}^2 \cdot \boldsymbol{Adv}^{\mathsf{Gap-aSDH}}_{\mathcal{B}_3^{\Phi_{\mathcal{G}}}}(\lambda)$$

where l_{H_i} denotes the number of H_i queries made by $\mathcal{A}$ for $i = 1, \ldots, 6$, p is the order of the finite field and l_{ssid} is the maximum number of subsessions between S *and* uid.

Proof Sketch. We describe the behaviour of the simulator based on the different corruption settings. The client can be either honest or corrupt while the server can be honest, corrupt or compromised. If none of the parties are corrupt, we further distinguish whether all message flows are oracle-generated or not. A flow is considered *oracle-generated* if it is sent by an honest party and arrives unaltered to the intended party. The simulator's behavior is described in Fig. 5 and Fig. 6. The different corruption settings are outlined in Table 1 in Appendix F. Skipping the case where both parties are corrupt, we start with the setting of Theorem 1 and introduce with each case new corruption settings. We outline the simulator behavior and indistinguishability arguments for each case. The full proof is given in Appendix F.

Case 2 + 3: Honest Client – Honest/Compromised Server – all flows oracle-generated: This setting is identical to the passive attacker model used in Theorem 1. Therefore, the main part of the simulation is as in the proof of Theorem 1. SRP-6a additionally uses the scrambling parameters u and k, resulting in two differences to the proof of Theorem 1. First, the intermediary session key in SRP-6a is $T_{\mathsf{C}} := g^{ab+bux}$ instead of $T_{\mathsf{C}} := g^{ab+bx}$, but since the simulator knows u, the same reduction argument from the proof of Theorem 1 also holds for SRP-6a. Second, the simulator aborts if he chooses B such that $B = kv$ (instead of $B = v$) for a compromised (s, v). Following the argumentation of Theorem 1, it follows that this simulation is indistinguishable under the Gap CDH assumption. Here, the DDH oracle as a global helper necessitates the use of the Gap CDH assumption.

Case 4: Corrupt Client – Honest Server: In this case, the simulator has to simulate the honest server's protocol messages without knowing the server's password. However, the simulator is allowed to guess one password pw^* per protocol session via a $\mathsf{TestPwd}$ query to $\mathcal{F}_{\mathsf{aPAKE}}$. If his guess is correct, the simulator can determine the session key of the server through a NewKey query. The simulation strategy is thus, to extract the corrupt client's password from the client's messages and check whether it matches the server's password.

The simulator is able to extract a password guess using the DDH oracle and his ability to manage the random oracle queries. After receiving message

Login Phase

(S1) On $(\mathsf{SvrSession}, \mathsf{ssid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ from $\mathcal{F}$:
(G_1) retrieve $\mathsf{file}[\mathsf{uid}] = (s, v)$. If $\nexists\ \mathsf{file}[\mathsf{uid}]$ do nothing and halt
(G_7) pick $B \xleftarrow{r} \mathbb{F}_p$
(G_7) if $(\mathsf{S}, \mathsf{uid})$ is marked compromised and $B = kv$ for $k := H_5(p, g)$, abort
(G_1) send (s, B) to $\mathcal{A}$ intended for C, keep record $(\mathsf{ssid}, \mathsf{S}, s, B)$.

(C2) On $(\mathsf{CltSession}, \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{F}$ and upon receiving (s, B) from $\mathcal{A}$:
(G_3) pick $A \xleftarrow{r} \mathbb{F}_p^*$, $M_1^{\mathsf{C}} \xleftarrow{r} \mathbb{F}_p$, compute $k := H_5(p, g)$
(G_1) send (A, M_1^{C}) to $\mathcal{A}$ intended for S, keep record $(\mathsf{ssid}, \mathsf{C}, s, B, k, A, M_1^{\mathsf{C}})$.

(S3) Upon receiving (A, M_1^{C}) from $\mathcal{A}$:
(G_3) if $\exists$ record $(\mathsf{ssid}, \mathsf{C}, s, B, k, A, M_1^{\mathsf{C}}) \wedge (\mathsf{S}, s, B)$ send $(\mathsf{NewKey}, \mathsf{ssid}, \mathsf{S}, \top)$ to $\mathcal{F}$
(G_3) upon receiving public delayed output $(\mathsf{ssid}, \bot)$, set $M_2^{\mathsf{S}} := \bot$
(G_3) else set $M_2^{\mathsf{S}} \xleftarrow{r} \mathbb{F}_p$
(G_5) if $\exists$ record $(\mathsf{impatt}, \mathsf{ssid}, *, A, M_1^{\mathsf{C}})$ abort
(G_7) if $\exists$ record $(\mathsf{guess}, \mathsf{ssid}, pw, T, A, M_1^{\mathsf{C}})$ send $(\mathsf{TestPwd}, \mathsf{ssid}, \mathsf{S}, pw)$ to $\mathcal{F}$:
(G_7) upon answer "correct guess":
(G_7) set $M_2^{\mathsf{S}} := H_3(A, M_1^{\mathsf{C}}, T)$, $K_{\mathsf{S}} := H_4(T)$
(G_7) upon answer "wrong guess":
(G_7) set $M_2^{\mathsf{S}} := \bot$, $K_{\mathsf{S}} := \bot$
(G_7) else set $(M_2^{\mathsf{S}}, K_{\mathsf{S}}) := (\bot, \bot)$
(G_{10}) send $(\mathsf{NewKey}, \mathsf{ssid}, \mathsf{S}, K_{\mathsf{S}})$ to $\mathcal{F}$
(G_1) send M_2^{S} to $\mathcal{A}$ intended for C.

(C4) Upon receiving M_2^{S} from $\mathcal{A}$:
(G_8) Retrieve record $(\mathsf{ssid}, \mathsf{C}, s, B, k, A, M_1^{\mathsf{C}})$
(G_8) If $\exists$ record $(H_3, [A, M_1^{\mathsf{C}}, T], M_2^{\mathsf{S}}) \wedge$ if $H_2(A, B, T) = M_1^{\mathsf{C}}$:
(G_8) set $K_{\mathsf{C}} := H_4(T)$
(G_8) else set $K_{\mathsf{C}} := \bot$
(G_{10}) send $(\mathsf{NewKey}, \mathsf{ssid}, C, K_{\mathsf{C}})$ to $\mathcal{F}$.

Fig. 5: Simulator SIM showing that SRP-6a UC-realizes $\mathcal{F} = \mathcal{F}_{\mathsf{aPAKE}}$, Part 1. The simulation of password registration, server compromise and the other random oracles are described in Fig. 3. Numbers indicate which game introduces a particular line of code. We assume that the simulator forwards all private delayed outputs to the corresponding parties.

(A, M_1^{C}), the simulator first extracts the adversary's ephemeral session key T_{C} from the random oracle query $H_2(A, B, T_{\mathsf{C}}) = M_1^{\mathsf{C}}$. If there is no such query, the simulator concludes that M_1^{C} is not constructed as in the protocol and simulates key exchange failure by setting $M_2^{\mathsf{S}} := \bot$. Recall here that H_2 is prefixed with sid and ssid which prevents the adversary from successfully passing off a M_1^{C} value obtained from some session ssid' as valid for session ssid. To see how a password can be extracted from T_{C}, consider that T_{C} is constructed as in the protocol for password pw^*. Then it holds that $T_{\mathsf{C}} = (B - kg^{x^*})^{a+ux^*} = \mathsf{cdh}_{\mathsf{g}}(Ag^{x^*u}, B - kg^{x^*})$ for $x^* = H_1(s, \mathsf{uid}, pw^*)$, $k = H_5(p, g)$ and $u = H_6(A, B)$. The simulator has

Simulation of H_2

On $H_2(A, B, T)$ from $\mathcal{A}$ or SIM:

(G_1) if this is the first such query, then
(G_4) if $\exists$ record $(H_5, [p, g], k)$, $(H_6, [A, B], u)$, $(H_1, [s, \mathsf{uid}, pw], x)$
(G_4) and $(H_1, [s, \mathsf{uid}, pw'], x')$ s.t. $\mathsf{ddh}_{\mathsf{g}}(Ag^{ux}, B - kg^{x}, T) = 1 \wedge$
(G_4) $\mathsf{ddh}_{\mathsf{g}}(Ag^{ux'}, B - kg^{x'}, T) = 1$, abort
(G_4) if $\exists$ record $(H_5, [p, g], k)$, $(H_6, [A, B], u)$, $(H_1, [s, \mathsf{uid}, pw], x)$
(G_4) s.t. $\mathsf{ddh}_{\mathsf{g}}(Ag^{ux}, B - kg^{x}, T) = 1$:
(G_6) if $\exists$ record $(\mathsf{guess}, \mathsf{ssid}, pw, T', *, *)$ with $T \neq T'$ abort
($G_{8/10}$) if $\exists$ record $(\mathsf{ssid}, \mathsf{C}, s, B, k, A, M_1^{\mathsf{C}})$ send $(\mathsf{TestPwd}, \mathsf{ssid}, \mathsf{C}, pw)$ to $\mathcal{F}$:
($G_{8/10}$) upon answer "correct guess": set $h := M_1^{\mathsf{C}}$
($G_{8/10}$) upon answer "wrong guess", set $h \xleftarrow{r} \mathbb{F}_p \setminus \{M_1^{\mathsf{C}}\}$
(G_4) else, pick $h \xleftarrow{r} \mathbb{F}_p$
(G_4) in any case, store record $(\mathsf{guess}, \mathsf{ssid}, pw, T, A, h)$
(G_4) else if $\exists$ record $(H_5, [p, g], k)$, $(H_6, [A, B], u)$ and compromised $(\mathsf{S}, \mathsf{uid})$
(G_4) with $\mathsf{file}[\mathsf{uid}] = (s, v)$ s.t. $\mathsf{ddh}_{\mathsf{g}}(Av^{u}, B - kv, T) = 1$:
($G_{8/10}$) if $\exists$ record $(\mathsf{ssid}, \mathsf{C}, s, B, k, A, M_1^{\mathsf{C}})$ send $(\mathsf{Impersonate}, \mathsf{ssid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}$:
($G_{8/10}$) upon answer "correct guess", set $h := M_1^{\mathsf{C}}$
($G_{8/10}$) upon answer "wrong guess", set $h \xleftarrow{r} \mathbb{F}_p \setminus \{M_1^{\mathsf{C}}\}$
(G_4) else, pick $h \xleftarrow{r} \mathbb{F}_p$
(G_4) in any case, store record $(\mathsf{impatt}, T, A, h)$
(G_1) else if no such records are found set $h \xleftarrow{r} \mathbb{F}_p$
(G_2) if $\exists$ record $(H_2, *, *, *, *, h)$ abort
(G_1) store $(H_2, [A, B, T], h)$ and reply with h
(G_1) else retrieve record $(H_2, [A, B, T], h)$ and reply with h

Fig. 6: Simulator SIM showing that SRP-6a UC-realizes $\mathcal{F} = \mathcal{F}_{\mathsf{aPAKE}}$, Part 2.

access to all these random oracle outputs and uses the DDH oracle to check if $\mathsf{ddh}_{\mathsf{g}}(Ag^{x^* u}, B - kg^{x^*}, T_{\mathsf{C}}) = 1$ for any combination of them. If this is the case, the simulator is able to extract the corrupt client's password from the H_1 query.

Using the $\mathsf{TestPwd}$ interface of $\mathcal{F}$, the simulator tests the extracted password against the honest server's password. If the passwords match, the simulator is able to continue the simulation with $T_{\mathsf{C}} = T_{\mathsf{S}}$, otherwise he simulates key exchange failure. This simulation is indistinguishable for the environment unless the adversary is able to guess the output of the random oracle or T_{C} corresponds to two password guesses. Both of these events only happen with negligible probability.

Case 5: Honest Client – Corrupt Server: The simulator in this case has to simulate the client's protocol messages without knowing his password. Again the $\mathsf{TestPwd}$ interface of $\mathcal{F}_{\mathsf{aPAKE}}$ allows the simulator to test one password per protocol run against the honest client's password. The simulation becomes more complex since the client is the first one to commit to a value based on the password. Hence, the simulator provides a non-committing message (A, M_1^{C}) which can later be "backpatched" to match the client's actual password if both

parties use the same password. To achieve this, the simulator uses his ability to control the random oracle. The corrupt server's password is extracted either from the messages received by the honest client or from the input to the random oracles. In detail, this case is handled as follows:

In (C2), the simulator uses random values for (A, M_1^{C}) and ensures that queries to H_2 are consistent with these values. An adversarial password guess relates to a query (A, B, T_{S}) to H_2 such that T_{S} is constructed as in the protocol for password pw^*. With the same argument as in Case 4, the simulator can extract the corrupt server's password by scanning all RO-outputs and using the DDH oracle to check whether $\mathsf{ddh}_{\mathsf{g}}(Ag^{x^*u}, B - kg^{x^*}, T_{\mathsf{S}}) = 1$. Using the $\mathsf{TestPwd}$ interface of $\mathcal{F}$, the simulator checks the extracted password against the server's password. If the passwords match, he programs the random oracle output of query (A, B, T_{S}) to match the previously generated M_1^{C} and can continue simulation with $T_{\mathsf{C}} = T_{\mathsf{S}}$. Since the corrupt server could skip the test whether $M_1^{\mathsf{C}} = H_2(A, B, T_{\mathsf{S}})$ in (S3), there might not be such a query to H_2 from the adversary. In this case, the simulator extracts T_{S} from message $M_2 = H_3(A, M_1^{\mathsf{C}}, T_{\mathsf{S}})$ and queries H_2 himself on (A, B, T_{S}). If there is no such query to H_3 the simulator concludes that M_2^{S} is not constructed as per the protocol and simulates key exchange failure. Similar to case 4, recall that H_3 is prefixed with sid and ssid, which prevents the adversary from passing off a M_2^{S} value from a session ssid' as valid for ssid.

This simulation is indistinguishable from the real world unless the adversary is able to query H_2 on two values $T_{\mathsf{S}}, T'_{\mathsf{S}}$ such that both of these values correspond to a password guess. In this case, the simulator can only test the password against the client's password for the first such query since he is limited to one password guess per protocol run. In this case, however, the adversary would have provided $(B, T_{\mathsf{S}}, T'_{\mathsf{S}})$ such that $\mathsf{ddh}_{\mathsf{g}}(A, B - kg^{x_1}, T_{\mathsf{S}}) = \mathsf{ddh}_{\mathsf{g}}(A, B - kg^{x_2}, T'_{\mathsf{S}}) = 1$ for two H_1 outputs x_1, x_2. If the Gap aSDH assumption holds, this cannot happen. Recall that we can set $B - kg^{x_1} - B - kg^{x_2} = kg^{x_1} - kg^{x_2} := \Delta$ and the adversary can embed aSDH challenge value Δ into the protocol by programming RO-output $H_5(p, g) = k := \Delta/(g^{x_1} - g^{x_2})$.

Case 6: Honest Client – Honest Server – not all flows oracle generated: In this scenario, the simulator has to simulate the protocol between two honest parties, while considering the possibility of a Man-in-the-Middle attack by an adversary who may inject his own messages. If the simulator receives a message on behalf of an honest party that was not oracle-generated, the simulator considers it an adversarial message and proceeds with the simulation as outlined in case 4 or 5.

Case 7: Corrupt Client – Compromised Server: This setting is identical to the setting of case 4, only that the server is now compromised and the adversary may know the password file. The simulation is handled exactly as in case 4. For indistinguishability, we have to argue that knowing the password verifier, but not the password itself does not allow the adversary to establish a key on the client's behalf.

Here, the challenge value $u = H_6(A, B)$ forces the client to prove that he can calculate a CDH tuple $\mathsf{cdh}_\mathsf{g}(v, B - kv)$ for password verifier $v := g^x$. Using the Forking Lemma [12], we can show that an adversarial client who is able to compute the ephemeral session key T_C without querying H_1 on the correct password, i.e. without knowing x, can be used to break a CDH challenge. In the reduction, after the adversarial client has computed $T_1 = \mathsf{cdh}_\mathsf{g}(Av^{u_1}, B - kv)$ for $u_1 = H_6(A, B)$, the CDH adversary rewinds him and changes the random oracle output to $u_2 = H_6(A, B)$ with $u_2 \neq u_1$. In this alternative execution, he obtains $T_2 = \mathsf{cdh}_\mathsf{g}(Av^{u_2}, B - kv)$. Using these values the adversary can embed his CDH challenge (g, g^d, g^e) by setting $v := g^d$, $B - kv := g^e$ and output $T_1/T_2^{1/(u_1-u_2)} = \mathsf{cdh}_\mathsf{g}(v, B - kv) = g^{de}$. Note that this rewinding is only used as an argument in the security proof, and the simulation of the protocol does not rely on rewinding adversaries which would be incompatible with security in the UC framework.

Case 8: Honest Client – Compromised Server – not all flows oracle generated: This setting is identical to case 6 with the added requirement for the simulator to handle server impersonation attempts. An adversary may have compromised the server and use the stolen password file to establish a key with the honest client by injecting his messages. The simulator can detect an impersonation attempt in a similar manner to detecting a password guess in case 5 by checking queries to the random oracle H_2. An adversarial impersonation attempt is indicated by a query (A, B, T_S) to H_2 such that $\mathsf{ddh}_\mathsf{g}(Av, B - kv, T_\mathsf{S}) = 1$ for RO-outputs k, u and stolen password verifier v. Whenever the simulator detects such a query he sends an $\mathsf{Impersonate}$ query to $\mathcal{F}$, which informs him whether the client runs on the same password. If this is the case, he backpatches the output of the H_2 query as outlined in case 5 and continues simulation with T_S. This simulation is indistinguishable from the real execution.

6.3 Discussion of Assumptions

In this subsection, we discuss our approach of using a DDH oracle within the angel-based UC framework and introducing the new aSDH assumption. We also discuss the implications and limitations of our approach and what can be shown without these assumptions.

DDH Oracles in Other PAKE Protocols. In the analysis of previous PAKE protocols [3,37], the existence of a DDH oracle was utilized, albeit with a slight distinction. These protocols typically required the existence of a DDH oracle within a reduction employed in the proof, whereas in our case, we necessitate a DDH oracle accessible to the simulator for protocol simulation. Our approach aligns with the angel-based UC framework, whereas the previous approach relied on the greater power of the adversary within the reduction. A similar consideration applies to the rewinding technique, which is disallowed for protocol simulation but can be utilized in a reduction argument presented in the proof. Our approach therefore differs from previous security analysis of PAKE protocols and we discuss the impact of our security proof.

Implications of Angel-based UC Security. In the angel-based UC framework, composability is only guaranteed when all protocols involved have access to the same angel. In the case of our SRP-6a protocol, its secure composition is therefore limited to protocols that remain secure in the presence of a DDH oracle in the finite field $\mathbb{F}_p$. Consequently, it cannot be securely composed with protocols that rely on the hardness of the DDH assumption in $\mathbb{F}_p$. Nevertheless, this is not a significant constraint since it is well known that the DDH assumption does not hold in $\mathbb{F}_p$ due to attacks exploiting the Legendre symbol [19,29].

The Legendre symbol characterizes whether an integer a is a quadratic residue modulo an odd prime p and is essentially a homomorphism from group elements to $\{-1, 1\}$. It possesses two crucial properties that allow it to break the DDH assumption: (a) It can be computed efficiently, and (b) it is multiplicative. When a DDH distinguisher receives a tuple (g^a, g^b, C), he can compute the Legendre symbol of g^a, g^b and C. Since the Legendre symbol of the primitive element g is -1, the Legendre symbols of g^a and g^b leak the parity of a and b. If C is chosen uniformly at random in the group, then with probability of $1/2$ its Legendre symbol does not match the parities of a and b. In this case, the adversary can successfully determine that $C \neq g^{ab}$. Consequently, the adversary's success probability is non-negligible, breaking the DDH assumption.

While this attack shows that the DDH assumption does not hold in $\mathbb{F}_p$, it does not imply the existence of an efficient DDH oracle for *all* values. Using the Legendre symbol technique, the DDH challenge can only be solved in case of *mismatching* parities. If the parities of a, b and C match (which occurs with probability $1/2$), the attacker has no advantage in deciding whether $C = g^{ab}$.

Thus, assuming the existence of an efficient DDH oracle gives the adversary and simulator an additional capability that captures the absence of the DDH assumption but also goes beyond what is currently known to be efficiently computable. We note that a similar approach is often used in security proofs of BLS type-3 pairings (see e.g. [21,30,64]) where it is sometimes assumed that the adversary has access to an isomorphism oracle that computes an isomorphism which is not efficiently computable.

Further, we emphasize again that the DDH oracle utilized in our proof is limited to the finite field $\mathbb{F}_p$ employed by the SRP protocol. This DDH oracle cannot be used to break DDH assumptions in other groups where the DDH assumption is believed to hold. In conclusion, we believe that the angel-based UC security of the SRP-6a protocol provides valuable composability guarantees.

Security without Angel. The simulator uses the DDH oracle as a superpolynomial time helper in order to extract adversarial password guesses. Without access to such an oracle, our proof only works in a setting where the client is always assumed to be honest. Thus, our proof would still allow the server to be actively corrupted and compromised, but the client needs to be honest. Note that for extracting adversarial password guesses from the server, no external DDH oracle is necessary: the simulator knows the ephemeral key a of $A = g^a$ and can use this value to efficiently test for $\mathsf{ddh}_{\mathsf{g}}(g^a g^{xu}, B - kg^x, T)$ for the server's queries (A, B, T) to H_2, thereby finding the password used by the

malicious server. In this case, also the strong aSDH assumption would suffice, since the simulator can use a restricted DDH oracle to simulate the client after embedding the aSDH challenge into the protocol. Such a partial DDH oracle that allows to test for DDH tuples w.r.t. a fixed X has been added to several DL-based assumptions [4,5] and can be implemented efficiently if x for $X = g^x$ is known.

An alternative method of weakening the security model is to use an ideal functionality which allows for late password tests [3], i.e., allowing the simulator to submit the extracted password after a session terminates. However, this is not useful in the SRP protocol, as the late password trick is only helpful when the client, from which we need to extract the password, makes another (observable) computation that depends on his password and which happens after receiving the server's message. Since this is not the case in SRP, there is no benefit in waiting and providing a late password interface.

Another idea to avoid the DDH oracle is to analyse SRP in a game-based security model for aPAKE. As the oracle is merely needed to extract passwords of malicious clients – such extractability is one of the core technical subtleties imposed by most UC functionalities – we expect that a security proof that does not require extractability also does not need permanent access to a DDH oracle. We opted for an analysis in the UC framework, as this has become the defacto standard for (a)PAKE protocols, and we believe that our security proof still provides sufficient insights into the UC security of SRP.

The aSDH Assumption. It is not very surprising that the SRP protocol needs a tailored assumption, based on its unique design choices and given the historical context of its development when very few cryptographic hardness assumptions had been established. Historically, all security proofs of protocols that were designed for a new kind of mathematical structure, required new and tailored assumptions such as the DH [33], RSA [55] and LRSW [51] assumptions. Currently, we are also unaware of other protocols which rely on this assumption. However, we strongly believe in the benefit of reducing the security of the SRP protocol to a simple mathematical assumption. The security now relies on a static and clean assumption that can be studied and analyzed, which is significantly simpler than analyzing the UC security of an interactive protocol against active adversaries from scratch.

Without the aSDH assumption (but with access to an DDH oracle), the proof still holds if we assume the server to be honest, in which case no two password guesses for one protocol run can occur.

UC Artifacts and Real-World SRP. Our analysis covers a UC-fied version of the SRP protocol which includes (sub)session identifiers sid and ssid. Here, we discuss the real-world mechanisms by which parties establish these values, and how security is impacted when they are missing.

First, recall that the session identifier sid is a static value tied to a server instance S in the form of $\mathsf{sid} = (\mathsf{S}, \mathsf{sid}')$ for a unique sid'. It is used to distinguish separate protocol instances and must be known to all users. In fact, a technical

minor – yet practically important – change we made in the aPAKE definition, is that we model a multi-user functionality where a single sid is shared by all users. This allows a more realistic setup and distribution of this value, compared to previous works where every user had to know and remember their unique sid: the sid can now be part of the server's certificate or part of the distributed protocol implementation.

The real-world SRP protocol does not have such an sid though. Protocols that don't use the sid no longer benefit from the composability guarantees of UC. However, the security properties of the stand-alone protocol are still guaranteed – at the same level as a security proof for a game-based property that does not provide any composability guarantees either.

The lack of an sid is nothing specific to the SRP protocol or our analysis, but a general gap between theory and practice of UC-secure protocols. For instance, OPAQUE [47] the state-of-the-art aPAKE protocol which is proven secure in the UC model, omits the sid in the practical implementation as described in the IRTF draft [49].

The second UC artifact is the sub-session identifier ssid that is assumed to be a unique input used to distinguish multiple login-sessions. A common method to establish an ssid in practice, is to let the client and server generate fresh nonces, and use the concatenation of both values as ssid (see e.g. [11,38]). As the UC definition requires that keys established through aPAKE are strictly bound to such an ssid, we include this identifier as prefix to all hash computations where session keys are derived. Without this prefix, an adversary could re-use key contributions from a session ssid in another session ssid$'$.

In the real-world protocol there are no sub-session identifiers, and consequently there is no need for such a hash prefix either. The session key in the standard SRP protocol is derived by hashing the exchanged values A, B along with the locally computed DH-value. This combination will be unique in every session where at least one of the parties is honest. Thus, even without the ssid, the established keys are bound to the particular session uniquely identified through the protocol transcript.

Again, this ssid-matter is nothing specific to the SRP-protocol or our analysis, but a general UC artifact. For instance, while the provably-secure version of OPAQUE uses ssids and the same hash prefixing approach to bind keys to ssids, the real-world specification omits both [49].

Overall, this might bear the question why we used the UC model for our analysis in the first place. We strongly believe that despite these artifacts, the UC functionality is the best available security model to analyze an aPAKE protocol such as SRP. The main benefit is that it naturally captures leakage, distribution bias and re-use of passwords, which a game-based definition can not reflect properly.

References

1. IEEE Standard specification for password-based public-key cryptographic techniques. IEEE Std 1363.2-2008

2. SRP stanford webpage. http://srp.stanford.edu
3. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. CRYPTO '20
4. Abdalla, M., Bellare, M., Rogaway, P.: The oracle diffie-hellman assumptions and an analysis of DHIES. CT-RSA '01
5. Abdalla, M., Haase, B., Hesse, J.: Security analysis of CPace. ASIACRYPT '21
6. Abdalla, M., Pointcheval, D.: Interactive diffie-hellman assumptions with applications to password-based authentication. FC '05
7. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. CT-RSA '05
8. Aggarwal, D., Maurer, U.: Breaking RSA generically is equivalent to factoring. EUROCRYPT '09
9. Apple: Escrow security for iCloud keychain (May 2021), https://support.apple.com/guide/security/escrow-security-for-icloud-keychain-sec3e341e75d/web
10. Apple: HomeKit communication security (May 2022), https://support.apple.com/guide/security/communication-security-sec3a881ccb1/web
11. Barak, B., Lindell, Y., Rabin, T.: Protocol initialization for the framework of universal composability. Cryptology ePrint Archive (2004)
12. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. ACM CCS '06
13. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. EUROCRYPT '00
14. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. ACM CCS '93
15. Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. ACM CCS '93
16. Bellovin, S.M., Merritt, M.: Cryptographic protocol for secure communications, US Patent 5241599A (Aug 1993)
17. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks IEEE SP '92
18. Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: New models and constructions. Cryptology ePrint Archive (2013)
19. Boneh, D.: The decision diffie-hellman problem. ANTS '98
20. Boneh, D., Lipton, R.J.: Algorithms for black-box fields and their application to cryptography. CRYPTO '96
21. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. ASIACRYPT '01
22. Boyko, V., MacKenzie, P., Patel, S.: Provably secure password-authenticated key exchange using diffie-hellman. EUROCRYPT '00
23. Butler, B.: Improved authentication for email encryption and security (Aug 2021), https://protonmail.com/blog/encrypted_email_authentication/
24. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. IEEE FOCS '01
25. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. EUROCRYPT '03
26. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. FOCS '10
27. Canetti, R., Rabin, T.: Universal composition with joint state. CRYPTO '03

28. Cash, D., Kiltz, E., Shoup, V.: The twin diffie-hellman problem and applications EUROCRYPT '09
29. Castryck, W., Sotáková, J., Vercauteren, F.: Breaking the decisional diffie–hellman problem for class group actions using genus theory: Extended version Journal of Cryptology, vol. 35, '22
30. Chatterjee, S., Hankerson, D., Knapp, E., Menezes, A.: Comparing two pairing-based aggregate signature schemes. Designs, Codes and Cryptography **55**
31. De Almeida Braga, D., Fouque, P.A., Sabt, M.: PARASITE: PAssword Recovery Attack against Srp Implementations in ThE wild. ACM CCS '21
32. Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols Communications of the ACM vol. 24, 1981
33. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions On Information Theory **22**(6) (1976)
34. Dos Santos, B.F., Gu, Y., Jarecki, S., Krawczyk, H.: Asymmetric PAKE with low computation and communication. EUROCRYPT '22
35. Fillion, R.: Developers: How we use SRP, and you can too: 1Password. `https://blog.1password.com/developers-how-we-use-srp-and-you-can-too/` (Feb 2018)
36. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. CRYPTO '06
37. Gu, Y., Jarecki, S., Krawczyk, H.: KHAPE: asymmetric PAKE from key-hiding key exchange. CRYPTO '21
38. Haase, B., Labrique, B.: AuCPace: Efficient verifier-based pake protocol tailored for the IIot TCHES '19
39. Hao, F.: On small subgroup non-confinement attack. IEEE CIT '10
40. Hao, F., van Oorschot, P.C.: SoK: Password-authenticated key exchange–theory, practice, standardization and real-world lessons. Cryptology ePrint Archive (2021)
41. Hesse, J.: Separating symmetric and asymmetric password-authenticated key exchange. SCN '20
42. ING Bank: Insidebusiness app security. `https://www.ingwb.com/en/service/insidebusiness-app/insidebusiness-app-security`
43. ISO: Information technology - security techniques - key management - part 4: Mechanisms based on weak secrets. ISO 11770-4:2006, Geneva, Switzerland (2006)
44. Jablon, D.P.: Cryptographic methods for remote authentication, US Patent 6792533 (May 2001)
45. Jager, T., Schwenk, J.: On the analysis of cryptographic assumptions in the generic ring model Journal of cryptology, vol. 26, '13
46. Jager, T., Schwenk, J.: On the equivalence of generic group models. ProvSec '08
47. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric pake protocol secure against pre-computation attacks. EUROCRYPT '18
48. Kidron, D., Lindell, Y.: Impossibility results for universal composability in public-key models and with fixed inputs Journal of Cryptology, vol.24, '11
49. Krawczyk, H., Bourdrez, D., Lewi, K., Wood, C.: The opaque asymmetric pake protocol, draft-irtf-cfrg-opaque-11 (2023)
50. Lopez, J., Dahab, R.: An overview of elliptic curve cryptography (2000)
51. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. SAC '99
52. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. EUROCRYPT '03
53. Pohlig, S., Hellman, M.: An improved algorithm for computing logarithms over GF(P) and its cryptographic significance. IEEE Transactions on Information Theory

54. Prabhakaran, M., Sahai, A.: New notions of security: Achieving universal composability without trusted setup. STOC '04
55. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems Communications of the ACM vol. 21, 1978
56. Roose, H.: Cognito. https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-authentication-flow.html
57. de Saint Guilhem, C.D., Fischlin, M., Warinschi, B.: Authentication in key-exchange: Definitions, relations and composition. IEEE CSF '20
58. Schmidt, J.: Requirements for password-authenticated key agreement (PAKE) schemes. RFC 8125 (Apr 2017)
59. SecureSafe: Securesafe homepage. https://www.securesafe.com/en/home (April 2023)
60. Shekh-Yusef, R., Sheffer, Y.: HTTP Secure remote password (SRP) authentication scheme. Internet-Draft draft-yusef-httpauth-srp-scheme-02, IETF (Jan 2016), https://datatracker.ietf.org/doc/html/draft-yusef-httpauth-srp-scheme-02, work in Progress
61. Sherman, A.T., Lanus, E., Liskov, M., Zieglar, E., Chang, R., Golaszewski, E., Wnuk-Fink, R., Bonyadi, C.J., Yaksetig, M., Blumenfeld, I.: Formal methods analysis of the secure remote password protocol. Logic, Language, and Security, '20
62. Shoup, V.: Lower bounds for discrete logarithms and related problems. EUROCRYPT '97
63. Shoup, V.: Security analysis of SPAKE2+. TCC '20
64. Smart, N.P., Vercauteren, F.: On computable isomorphisms in efficient asymmetric pairing-based systems Discrete Applied Mathematics 155
65. Szydlo, M.: A note on chosen-basis decisional diffie-hellman assumptions. FC '06
66. Taylor, D., Perrin, T., Wu, T., Mavrogiannopoulos, N.: Using the secure remote password (SRP) protocol for TLS authentication. RFC 5054 (Nov 2007)
67. TeamViewer: Security meets usability: The teamviewer revolution. https://www.teamviewer.com/en-us/security-meets-usability-the-teamviewer-revolution/ (April 2023)
68. Telegram: Two-factor authentication (April 2023), https://core.telegram.org/api/srp
69. Wikipedia: Secure remote password protocol. https://en.wikipedia.org/wiki/Secure_Remote_Password_protocol (Jan 2022)
70. Wong, D.: User authentication with passwords, what's SRP? https://www.cryptologie.net/article/503/user-authentication-with-passwords-whats-srp/ (May 2020)
71. Wu, T.: SRP-6: Improvements and refinements to the secure remote password protocol. http://srp. stanford. edu/srp6.ps (2002)
72. Wu, T.: The SRP authentication and key exchange system. RFC 2945 (Sep 2000)
73. Wu, T.D.: The secure remote password protocol. NDSS '98
74. Yen, A.: Proton pass is now in beta (April 2023), https://proton.me/blog/proton-pass-beta

A Formal Definitions for Assumptions in $\mathbb{F}_p$

Here, we give a formal definition of the CDH, DDH and Gap CDH assumption in finite fields.

Definition 3 (Computational Diffie Hellman Assumption over $\mathbb{F}_p$ (CDH)). *We say that the CDH problem is hard relative to $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$,*

$$\mathbf{Adv}_{\mathcal{A}}^{CDH}(\lambda) := \Pr[\mathcal{A}(\mathbb{F}_p, p, g, g^a, g^b) = g^{ab}]$$

is negligible in λ, where $(\mathbb{F}_p, p, g) \xleftarrow{r} \mathcal{G}(1^\lambda), a \xleftarrow{r} \mathbb{Z}_{p-1}, b \xleftarrow{r} \mathbb{Z}_{p-1}$.

Definition 4 (Decisional Diffie Hellman Assumption over $\mathbb{F}_p$ (DDH)). *We say that the DDH problem is hard relative to $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$*

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{DDH}(\lambda) :=& |\Pr[\mathcal{A}(\mathbb{F}_p, p, g, g^a, g^b, g^{ab}) = 1] \\ &- \Pr[\mathcal{A}(\mathbb{F}_p, p, g, g^a, g^b, g^c) = 1]|\end{aligned}$$

is negligible in λ, where $(\mathbb{F}_p, p, g) \xleftarrow{r} \mathcal{G}(1^\lambda), a \xleftarrow{r} \mathbb{Z}_{p-1}, b \xleftarrow{r} \mathbb{Z}_{p-1}, c \xleftarrow{r} \mathbb{Z}_{p-1}$.

Definition 5 (Gap CDH Assumption over $\mathbb{F}_p$ (GapCDH)). *We say that the Gap CDH problem is hard relative to $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$*

$$\mathbf{Adv}_{\mathcal{A}}^{Gap\text{-}CDH}(\lambda) := \Pr[\mathcal{A}^{\mathsf{ddh}_{\mathsf{g}}(\cdot,\cdot,\cdot)}(\mathbb{F}_p, p, g, g^a, g^b) = g^{ab}]$$

is negligible in λ, where $(\mathbb{F}_p, p, g) \xleftarrow{r} \mathcal{G}(1^\lambda), a \xleftarrow{r} \mathbb{Z}_{p-1}, b \xleftarrow{r} \mathbb{Z}_{p-1}$ and $\mathsf{ddh}_{\mathsf{g}}(g^x, g^y, g^z) = 1 \Leftrightarrow xy = z$.

B Flawed Reduction in SRP-3

In [73], the author establishes the security of SRP against passive eavesdropping attacks by a reduction to the Computational Diffie Hellman problem. The reduction they provided is correct but it cannot be used to argue anything about the security of the SRP protocol.

They argue that if there exists an efficient adversary Q (in the paper described as oracle) that can compute the SRP session key given all public information and the user's password, then this adversary can be used to construct an adversary winning the CDH challenge. Their adversary for SRP accepts values A, B, u, g, p, and x and computes the intermediary session key $T = g^{ab+bux}$.

$$Q(g^a, g^b + g^x, u, g, p, x) = g^{ab+bux}$$

Then they claim that given a CDH challenge (g, g^a, g^b) they can solve it using adversary Q with inputs $u = 2$ and $x = (p-1)/2$. Thus, for the adversary Q' defined as

$$Q'(A, B, g, p) = Q\Big(A, B + g^{(q-1)/2}, 2, g, p, (p-1)/2\Big)$$

it holds that $Q'(g^a, g^b, g, q) = g^{ab}$ and the adversary wins the CDH game.

While this reduction works, it only rules out the existence of an efficient adversary Q who can compute the session key for hardcoded inputs $u = 2$ and $x = (p-1)/2$. This says nothing about the security of the SRP protocol. In

fact, an adversary who aborts after receiving value $x = (p-1)/2$ but outputs $K = g^{ab+bux}$ for any other value of x cannot be used for this reduction but clearly breaks the security of the SRP protocol. We can circumvent this problem by, after receiving CDH challenge g^a, g^b, querying Q on input $(g^a, g^b + g^x, u, g, p, x)$ for *randomly* sampled u, x and multiplying the output g^{ab+bux} of Q with $((g^b)^{ux})^{-1}$, obtaining g^{ab} and thus winning the CDH experiment. We use this modified reduction in our security proof of SRP_{base}.

C Preventing Client Impersonation Attack with Value u

As highlighted in [73], in SRP_{base} and SRP-1 it is possible to impersonate a client by knowing only the password verifier $v = g^x$. That is, after server compromise no offline attack on the password is needed in order to establish a common shared key with the server on the client's behalf. The client impersonation attack works as follows: In Fig. 2 (C2), the malicious client $\mathcal{A}_{\mathsf{C}}$ who knows the password verifier v, chooses a random a and sets $A := g^a v^{-1}$. This cancels out the "v term" in the server's session key calculation and the shared ephemeral key is $T_{\mathsf{S}} := (Av)^b = (g^a v^{-1} v)^b = g^{ab}$ which the client can compute without knowledge of $x = H_1(s, \mathsf{uid}, pw)$. Thus, no offline dictionary attack is needed to impersonate the client. This violates the security guarantees of the ideal functionality $\mathcal{F}_{\mathsf{aPAKE}}$ which states that a client must know the correct password in order to derive a shared key with the server.

To mitigate this attack, a "challenge" value u was introduced in SRP-3 [73] and the intermediary key is calculated as $T_{\mathsf{S}} := (Av^u)^b$, and $T_{\mathsf{C}} := (B - g^x)^{a+ux}$, respectively. This does not prevent the attack yet, as the adversary could now set $A := g^a v^{-u}$. To ensure that the adversary cannot tweak the A value in such a way, the challenge value u is computed by hashing the protocol messages A and B together: $u := H(A, B)$. Thus, in order to execute the same attack as before, the adversary must find a value u for which $u = H(g^a v^{-u}, B)$ holds. This explicit attack is infeasible if the hash function is preimage resistant. In order to rule out any kind of impersonation attack using a stolen verifier, we model the hash function as a random oracle in the security proof of Theorem 2 and relate the probability of an adversary performing a client impersonation attack to the hardness of the CDH problem in $\mathbb{F}_p^*$.

D On Proving Hardness of the aSDH Assumption

To show the soundness of the aSDH assumption, one would ideally relate this new assumption to existing ones or prove hardness in a generic model. In this section, we elaborate on the challenges for both of these approaches in regards to the aSDH assumption.

Relation to CCDH and CDH. As explained in the main body, the aSDH assumption is structurally similiar to the CCDH assumption proposed by Abdalla and Pointcheval [7]. They are able to show that the CDH and CCDH assumption are

equivalent using a reduction which utilizes the multiplicativity of the CDH tuple, namely $(Y/N)^x = Y^x/N^x$. However, when it comes to the aSDH assumption, a similar reduction cannot be applied due to the fact that $(Y + \Delta)^x \neq Y^x + \Delta^x$ with overwhelming probability for given values of Y, Δ, and x.

It is worth mentioning that a variant of the CCDH assumption, known as the Chosen-Basis DDH assumption [6], was later found to be vulnerable to attacks [65]. However, this does not undermine the validity of the original CCDH assumption, as the Chosen-Basis DDH assumption is a decisional variant that involves two rounds of interaction, rather than a computational assumption. Furthermore, the Chosen-Basis DDH assumption bears no resemblance to the aSDH assumption we are considering, and the attack proposed against it cannot be leveraged to break the aSDH assumption.

Hardness in a Generic Model. Another approach to show the validity of the aSDH assumption could involve showing the hardness in a generic model such as the black-box field model introduced by Boneh and Lipton [20]. In this model, the generic field is modeled as a black box where the field operations – addition, multiplication and equality tests – are performed by accessing an oracle. In the literature, there is no proof of hardness for any assumption in the generic field model, and Boneh and Lipton even show, albeit using an unproven number-theoretic assumption, that the Black Box Field Problem – the generic equivalent of the discrete logarithm problem – can be solved by an algorithm in sub-exponential time.

The next thought would be to study the hardness of the aSDH assumption in the generic ring model. Most of the research in this area such as [8,45] has focused on relating the hardness of a problem to factoring or using the generic model for reductions to other CDH type problems. Since the order of the ring contained in $\mathbb{F}_p$ is not related to factoring two large prime numbers and since there is no known assumption that we can reduce aSDH to, the known proof techniques cannot be used to show the hardness of the aSDH assumption.

The most promising research regarding generic algorithms has been in the generic group model (see e.g. [46,62]). However, since the aSDH assumption incorporates both addition and multiplication in the field, the generic group model is not valid for our case. Additionally, as pointed out in [45] the proof techniques from the generic group model are not applicable to proofs in the generic ring model.

We identify the analysis of hardness assumptions in a generic field model as future work. Although this is no proof of soundness, the SRP-6a protocol has been around for almost 20 years and there has been no evidence of a successful two-for-one guessing attack after the introduction of the value k. This further strengthens our believe in the soundness of the assumption.

E Proof of Theorem 1

In order to show that the protocol UC-realizes the functionality $\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$, we need to show that for all environments and all adversaries, we can construct a simula-

tor such that the interactions, from the one hand between the environment, the players (C and S) and the adversary (the real world), and from the other hand between the environment, the ideal functionality and the simulator (the ideal world), are indistinguishable for the environment. In this proof, we incrementally define a sequence of games starting with the real execution of the protocol and ending up with game 6, which we prove to be indistinguishable from the ideal experiment. Note that these games are sequential and built on each other. We only describe the changes that occur when changing from Game G_i to Game G_{i+1}. We assume that the rest of Game G_{i+1} behaves exactly as in Game G_i. Let $\text{Real}_{\mathcal{Z}}(\text{SRP}, \mathcal{A})$ be the event that environment $\mathcal{Z}$ with adversary $\mathcal{A}$ and an execution of SRP outputs 1 and $\text{Ideal}_{\mathcal{Z}}(\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}, \mathsf{SIM})$ be the corresponding event in the ideal execution with ideal functionality $\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$. Additionally in the analysis of the games let $\mathsf{G_i}$ denote the event that the environment $\mathcal{Z}$ outputs 1 in game G_i.

Game G_0: The real protocol execution. This is the real world in which the adversary may eavesdrop on messages sent by the real parties and obtain the password file stored by the server.

$$\Pr[\text{Real}_{\mathcal{Z}}(\text{SRP}, \mathcal{A})] = \Pr[\mathsf{G_0}]$$

Game G_1: Introducing the simulator. This game converts the real protocol into a simulated environment where the random oracles are implemented by lazy sampling.

Changes to the simulation: We move the whole execution of the protocol into one machine and call it the simulator SIM. For clarity, sometimes we write $\mathsf{SIM_S}$, whenever SIM acts on behalf of the server and $\mathsf{SIM_C}$ if SIM acts on behalf of the client. Note that SIM still runs the protocol execution with the actual passwords as input. In later iterations of the games the simulation will work without the passwords as input. We simulate the passive eavesdropping ability of the adversary by sending the protocol messages to $\mathcal{A}$ intended for the other party. The adversary has to relay them unchanged to the other party, otherwise the simulation aborts.

Changes to the random oracle: SIM implements the random oracles H_1, H_2, H_3 and H_4 by lazy sampling of the output values, i.e. every new query to H_i is answered with a randomly sampled value from the output domain of H_i.

Indistinguishability argument: The changes are only syntactical and thus, the games are indistinguishable.

$$\Pr[\mathsf{G_1}] = \Pr[\mathsf{G_0}]$$

Game G_2: Abort on collisions of the random oracles. In this game the simulator aborts on collisions of the random oracle. This ensures that two honest parties who follow the protocol but don't have matching passwords will output $(\mathsf{ssid}, \perp)$, whereas honest parties with matching passwords who follow the protocol will output the same session key $(\mathsf{ssid}, K_{\mathsf{C}}) = (\mathsf{ssid}, K_{\mathsf{S}})$.

Changes to the random oracle: The simulator aborts if a collision occurs in H_1, H_2, H_3 or H_4, i.e. if SIM samples an answer for a fresh query that he already gave before.

Indistinguishability argument: An adversary can only distinguish the two games if an abort occurs. But, due to the birthday bound, the probability of an abort happening is negligible in λ. Let l_{H_i} denote the number of queries made to H_i for $i = 1, 2, 3, 4$. Then,

$$|\Pr[\mathsf{G}_2] - \Pr[\mathsf{G}_1]| \leq \frac{l_{H_1}^2}{p-1} + \frac{l_{H_2}^2}{p} + \frac{l_{H_3}^2}{p} + \frac{l_{H_4}^2}{2^\lambda}$$

Game G_3: Simulate all protocol values after (S1). In this game, we simulate the protocol messages after (S1) and the session keys using random values. This makes them independent of the passwords provided by the environment. After (S1) the simulation's only dependence on the password is the check whether the passwords match which determines whether the values $M_2^S, K_\mathsf{S}, K_\mathsf{C}$ are each a rejection symbol $\perp$ or sampled uniformly at random. This password check will be performed using the NewKey interface of the ideal functionality in a later game.

First, $\mathsf{SIM_C}$ sends a random $M_1^\mathsf{C} \xleftarrow{r} \mathbb{F}_p$ to $\mathsf{SIM_S}$, then $\mathsf{SIM_S}$ checks if the parties' passwords match and either sets $(M_2^\mathsf{S}, K) = (\perp, \perp)$ (if $pw \neq pw'$) or $M_2^\mathsf{S} \xleftarrow{r} \mathbb{F}_p, K \xleftarrow{r} \{0,1\}^\lambda$ (if $pw = pw'$). $\mathsf{SIM_S}$ sends M_2^S to $\mathsf{SIM_C}$ and outputs (ssid, K) to both parties. This change is only noticeable if the adversary can compute a CDH challenge.

Changes to the simulation: In (C2), $\mathsf{SIM_C}$ samples (A, M_1^C) uniformly at random from $\mathbb{F}_p^* \times \mathbb{F}_p$. Since this prevents $\mathsf{SIM_S}$ from checking whether $M_1^\mathsf{C} = M_1^\mathsf{S}$ in (S3), $\mathsf{SIM_S}$ introduces a password check to see if both parties' password match. In case of matching passwords, M_2^S and K_S are taken uniformly at random from $\mathbb{F}_p$ and $\{0,1\}^\lambda$, respectively, otherwise they are both set to $\perp$. In (C4), in case of matching passwords, $\mathsf{SIM_C}$ sets $K_\mathsf{C} := K_\mathsf{S}$, otherwise he sets $K_\mathsf{C} := \perp$.

Indistinguishability argument: The only way for an adversary to distinguish the two games is by noticing the difference in the creation of protocol values $A, M_1^\mathsf{C}, M_2^\mathsf{S}, K_\mathsf{C}$ and K_S. A is indistinguishable since we only removed the simulator's knowledge of a such that $A = g^a$. The other values are outputs of the random oracles H_2, H_3 and H_4 in game 2 while they are uniformly random values sampled from the same domain in game 3. Thus, in order to distinguish them, the adversary has to compute the value $T = \mathsf{cdh_g}(Ag^{ux}, B - kg^x)$ and query one of the random oracles on the correct input, i.e. H_2 on (A, B, T), H_3 on (A, M_1^C, T) or H_4 on T. Note that in case of mismatching passwords, the adversary can only distinguish the games using M_1^C since the other values are set to $\perp$. We show that the adversary can compute the value T only with negligible probability if the CDH assumption holds in $\mathbb{F}_p$ even if he knows the parties' secret inputs pw and pw'.

Therefore, we construct an efficient CDH adversary $\mathcal{B}_\mathsf{CDH}$ interacting with $\mathcal{Z}$. Let (g, g^d, g^e) denote a CDH challenge. $\mathcal{B}_\mathsf{CDH}$ embeds the challenge in the game as follows: $\mathcal{B}_\mathsf{CDH}$ samples $x' \xleftarrow{r} \mathbb{Z}_{p-1}$ and sets $(H_1, [s, \mathsf{uid}, pw'], x')$ where pw' is

the client's password. Note that $\mathcal{B}_{\mathsf{CDH}}$ receives both parties' passwords as input. Then, $\mathcal{B}_{\mathsf{CDH}}$ sets $A := g^d$ and $B := g^e + kg^{x'}$ for $k = H_5(p, g)$. Now we consider the two cases:

1. $pw = pw'$: If the passwords of both parties match, $\mathcal{B}_{\mathsf{CDH}}$ samples $s \xleftarrow{r} \{0,1\}^\lambda$, $M_1^{\mathsf{C}} \xleftarrow{r} \mathbb{F}_p$, $M_2^{\mathsf{S}} \xleftarrow{r} \mathbb{F}_p$, $K \xleftarrow{r} \{0,1\}^\lambda$ and simulates the protocol with transcript $(s, B, A, M_1^{\mathsf{C}}, M_2^{\mathsf{S}})$ and output (ssid, K) for both parties. Note that in this case it holds that $T_{\mathsf{S}} = T_{\mathsf{C}} = (B - kg^{x'})^{d+x'} = g^{de+ex'}$. Let ℓ_R denote the number of records of the form $(H_2, [A, B, *], *)$, $(H_3, [A, M_1^{\mathsf{S}}, *], *)$ or $(H_4, [*], *)$. Note that l_R is (polynomially) bounded by $l_{H_2} + l_{H_3} + l_{H_4}$. $\mathcal{B}_{\mathsf{CDH}}$ picks one of these records at random, extracts T^* from the corresponding query and outputs $T^* \cdot ((g^d)^x)^{-1}$. If $T^* = g^{de+dx}$, then $\mathcal{B}_{\mathsf{CDH}}$'s output equals g^{de} and $\mathcal{B}_{\mathsf{CDH}}$ wins in the CDH experiment.
2. $pw \neq pw'$: If the passwords of both parties don't match $\mathcal{B}_{\mathsf{CDH}}$ samples $s \xleftarrow{r} \{0,1\}^\lambda$, $M_1^{\mathsf{C}} \xleftarrow{r} \mathbb{F}_p$ and simulates the protocol with transcript $(s, B, A, M_1^{\mathsf{C}}, \perp)$ and output $(\mathsf{ssid}, \perp)$ for both parties. Again, in a real execution of the protocol C would compute $T_{\mathsf{C}} = (B - g^{x'})^{d+x'} = g^{de+ex'}$ and it would hold that $M_1^{\mathsf{C}} = H_2(A, B, T_{\mathsf{C}})$. Let l_{H_2} denote the number of records of the form $(H_2, [A, B, *], *)$. l_{H_2} is (polynomially) bounded by l_{H_2}. $\mathcal{B}_{\mathsf{CDH}}$ picks one of those record uniformly at random. Let this record be $(H_2, [A, B, T^*], M_1^*)$. $\mathcal{B}_{\mathsf{CDH}}$ outputs $T^* \cdot ((g^e)^{x'})^{-1}$. If $T^* = g^{de+ex'}$, then $\mathcal{B}_{\mathsf{CDH}}$'s output equals g^{de} and $\mathcal{B}_{\mathsf{CDH}}$ wins in the CDH experiment.

Thus, if an adversary successfully distinguishes the two games, he can win the CDH experiment with non negligible probability. We thus have

$$|\Pr[\mathsf{G_3}] - \Pr[\mathsf{G_2}]| \leq (l_{H_1} + 2l_{H_2} + l_{H_3}) \cdot \mathbf{Adv}^{\mathsf{CDH}}_{\mathcal{B}_{\mathsf{CDH}}}(\lambda)$$

Game G_4: Change B to be independent of the password file. In this game we make B independent of the password file v by picking a random field element for B. This change makes all protocol values independent of the parties' passwords.

Changes to the simulation: In (S4), B is computed by sampling $B \xleftarrow{r} \mathbb{F}_p$.

Indistinguishability Argument: No protocol value depends on B anymore so we only have to check that B is indistinguishable in both games. In game 3, B is chosen as $B := v + g^b$ for a uniformly random $b \xleftarrow{r} \mathbb{Z}_{p-1}$. Thus, B is a uniform random element from $\mathbb{F}_p \setminus \{v\}$. In game 4, B is a uniform random element from $\mathbb{F}_p$. The games are thus distinguishable if $B = v$ in game 4, especially since the adversary might know the server's password file. Since $v = g^x$ for the randomly sampled H_1-output x in game 3, the probability of an abort is negligible.

$$|\Pr[\mathsf{G_4}] - \Pr[\mathsf{G_3}]| \leq \frac{1}{p-1}$$

Recap of the changes so far: We recap the changes made to the protocol thus far. We have replaced all values of a protocol execution with random elements. Hence, no protocol value depends on the parties' secret inputs anymore.

The only dependence on the passwords is when creating a password file which will be removed next.

Game G_5: Change Password File Creation and StealPwdFile interaction. The password file is only completely generated upon a StealPwdFile query. For a login session only the client-specific salt is needed. When generating v for a stolen password file, the consistency of v is crucial. Therefore, after receiving a StealPwdFile query, if the correct password was queried to H_1 before, x is set to be the output of the corresponding query. Otherwise a random x is taken and $v = g^x$ sent to the adversary. In order to make sure that future queries to H_1 are consistent with x, if H_1 is queried for (s, uid, pw) on the client's salt s and the correct password pw after compromise, the output of this query is set to x.

Changes to the simulation: On the first call to either SvrSession or StealPwdFile from $\mathcal{F}$, $\mathsf{SIM_S}$ picks a random salt s and records $\mathsf{file}[\mathsf{uid}] = (s, \perp)$. The whole password file is only generated upon receiving query $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ as follows: $(\mathsf{S}, \mathsf{uid})$ is marked compromised and, if there is a record $(H_1, [s, \mathsf{uid}, pw^*], x)$ where s matches the salt in the password file, and $pw^* = pw$, then $\mathsf{SIM_S}$ retrieves x. Otherwise a random $x_{\mathsf{uid}} \xleftarrow{r} \mathbb{Z}_{p-1}$ is chosen. If $g^{x_{\mathsf{uid}}} = B_{\mathsf{S}}^{\mathsf{ssid}}$, that is, if the verifier matches the B value of a previous session, the simulator aborts. Otherwise, the password file (s, v) with $v = g^{x_{\mathsf{uid}}}$ is sent to the adversary.
Changes to the random oracles: H_1 is changed as follows: For every query (s, uid, pw) such that s matches the salt used for uid, $\mathsf{SIM_S}$ does the following: If S is marked compromised and the password is correct, sets $x = x_{\mathsf{uid}}$ where x_{uid} was used to simulate the password verifier upon compromise. This ensures the output of the random oracle query to be consistent with the password file sent to the adversary. If S is marked compromised and the password is not correct, the simulator sets $x \xleftarrow{r} \mathbb{Z}_{p-1} \setminus \{x_{\mathsf{uid}}\}$. In any other case a random x is chosen to answer the query.

Indistinguishability argument: If there is no abort, the games are indistinguishable. The probability of an abort is bounded by the maximum number of subsessions for the same party S. Let l_{ssid} denote this number. Thus it holds that

$$|\Pr[\mathsf{G_5}] - \Pr[\mathsf{G_4}]| \leq l_{\mathsf{ssid}} \frac{1}{p}$$

Game G_6: Introduce $\mathcal{F}$, password check is handled by $\mathcal{F}$ In this game, we introduce the functionality $\mathcal{F} = \mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$ and let SIM communicate with $\mathcal{F}$. The functionality handles the password check for online and offline sessions and sends the protocol output to the parties after being instructed to do so by SIM.

Changes to the simulation: SIM is triggered by receiving a CltSession or SvrSession message from $\mathcal{F}$ instead of receiving these messages directly from the parties. The simulator does not receive the parties' secret inputs anymore. The password check in (S3) and (C4) is replaced by a NewKey query to $\mathcal{F}$, which informs SIM if the passwords don't match by responding with a public delayed output $(\mathsf{ssid}, \perp)$. If the passwords match, a random key is generated and sent to the parties by $\mathcal{F}$ instead of SIM. After receiving a StealPwdFile query from $\mathcal{A}$,

SIM forwards this query to $\mathcal{F}$ who informs SIM if the password was queried to H_1 with the correct combination of s and uid before.

Changes to the random oracles: For every query (s, uid, pw) such that s matches the salt used for some S, SIM sends an $\mathsf{OfflineTestPwd}$ query to $\mathcal{F}$ who checks if $(\mathsf{S}, \mathsf{uid})$ is compromised and if the password matches the password of S. If SIM receives "correct guess", he sets the output of this query to be $x = x_{\mathsf{S}}^{\mathsf{uid}}$, if he receives "wrong guess", he sets the output of this query to be $x \neq x_{\mathsf{S}}^{\mathsf{uid}}$. In every other case, SIM picks a random element $x \xleftarrow{r} \mathbb{Z}_{p-1}$.

Changes to the functionality: The full functionality $\mathcal{F} = \mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$ is added.

Indistinguishability Argument: The changes are only internal with the main difference being in case of matching passwords where both players now receive a random key chosen by $\mathcal{F}$ instead of SIM. This is indistinguishable and thus,

$$\Pr[\mathsf{G}_6] = \Pr[\mathsf{G}_5] = \Pr[\mathrm{Ideal}_{\mathcal{Z}}(\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}, \mathsf{SIM})]$$

This game is identical to the ideal execution with $\mathcal{F}_{\mathsf{aPAKE}}^{\mathsf{passive}}$, which concludes the proof.

F Proof of Theorem 2

Table 1: The different corruption cases for the proof of Theorem 2. The client can either be honest or corrupt while the server can be honest, corrupt or compromised. In case none of the party is corrupted we distinct whether all flows are oracle-generated or not.

Case	Client	Server	all flows oracle-generated
(1)	corrupt	corrupt	–
2	honest	honest	yes
3	honest	compromised	yes
4	corrupt	honest	–
5	honest	corrupt	–
6	honest	honest	no
7	corrupt	compromised	–
8	honest	compromised	no

We start with the real execution of the SRP-6a protocol with an adversary $\mathcal{A}$ and gradually modify it, ending up with the ideal execution $\mathcal{F}_{\mathsf{aPAKE}}$ with simulator SIM. The changes will go unnoticed by an environment $\mathcal{Z}$ interacting with the parties and the adversary.

Let $\mathsf{Real}_{\mathcal{Z}}(\text{SRP-6a}, \mathcal{A})$ be the event that the environment $\mathcal{Z}$ with adversary $\mathcal{A}$ and an execution of SRP-6a outputs 1 and $\mathsf{Ideal}_{\mathcal{Z}}(\mathcal{F}_{\mathsf{aPAKE}}, \mathsf{SIM}^{\Phi_{\mathcal{G}}})$ be the corresponding event in the ideal execution with simulator SIM and functionality

$\mathcal{F}_{\mathsf{aPAKE}}$ depicted in Fig. 1. Additionally in the analysis of the games let $\mathsf{G_i}$ denote the event that the environment $\mathcal{Z}$ outputs 1 in game G_i.

Game G_0: The real protocol execution. This is the real world where the adversary interacts with real players and may (statically) corrupt parties. He may view, modify and drop messages sent by honest parties and send messages on behalf of corrupted parties. Additionally, the adversary is allowed to steal an honest server's password file.

$$\Pr[\mathsf{Real}_{\mathcal{Z}}(\text{SRP-6a}, \mathcal{A}^{\Phi_{\mathcal{G}}})] = \Pr[\mathsf{G_0}]$$

Game G_1: Introducing the simulator. In this game we move the whole execution into one machine and call it the simulator SIM. For clarity, in some cases we write $\mathsf{SIM_C}$ if the simulator simulates the behaviour of an honest client and $\mathsf{SIM_S}$ if he simulates an honest server's behaviour. We model the adversarial interference by sending honest party's messages to the adversary $\mathcal{A}$ intended for the other party. The adversary can alter the message before sending it to the other party or drop the message, i.e. not send the message at all. This game change also implies that SIM implements the random oracles $H_1 - H_6$ and runs the execution with actual passwords as input. We will change the simulation to work without passwords in the upcoming games. In this game the random oracles are simulated by lazy sampling of output values. That is, for any new input query to H_i $(i = 1, \ldots, 6,)$ SIM answers with a value chosen uniformly at random from the output domain $\mathcal{D}(H_i)$ of H_i. No re-programming operation is yet needed. The changes are only syntactical and thus

$$\Pr[\mathsf{G_0}] = \Pr[\mathsf{G_1}]$$

Game G_2: Abort on collisions of a random oracle. For $i = 1, \ldots, 6$, the simulator aborts if a collision occurs in H_i, i.e. if SIM samples an answer for a fresh H_i query that he already gave before. Note that without collisions two honest parties will always output matching (respectively differing) session keys if both, passwords and party identifiers, match (respectively differ). $\mathcal{Z}$ can only make a polynomially bounded number l_{H_i} queries to each hash function H_i and the size of the output domain is $p-1$ for H_1, H_5 and H_6, p for H_2 and H_3, and 2^λ for H_4. Since $p \approx 2^\lambda$, the probability of an abort is therefore negligible in λ by the birthday bound. Thus, G_2 is computationally indistinguishable from G_1:

$$|\Pr[\mathsf{G_2}] - \Pr[\mathsf{G_1}]| \leq \frac{l_{H_1}^2}{p-1} + \frac{l_{H_2}^2}{p} + \frac{l_{H_3}^2}{p} + \frac{l_{H_4}^2}{2^\lambda} + \frac{l_{H_5}^2}{p-1} + \frac{l_{H_6}^2}{p-1}$$

Game G_3: Simulate protocol execution between two honest parties. In case both parties are honest and no message gets altered by the adversary, the simulator uses random protocol values to simulate the protocol similar to the simulator in SRP_{base}.

Changes to the simulation: In (C2), if all flows are oracle-generated, the protocol messages A and M_1^{C} are sampled uniformly at random from $\mathbb{F}_p^*$ and $\mathbb{F}_p$, respectively. The simulator distinguishes oracle-generated messages from non-oracle-generated messages by keeping record of his simulated messages. In (S3),

if all flows are oracle-generated he performs a password check using his knowledge of both parties' private inputs. If their passwords match, he chooses M_2^{S} and K_{S} uniformly at random from the corresponding domains, otherwise he sets them both to $\perp$. In (C4), if $\mathsf{SIM}_{\mathsf{C}}$ receives an oracle-generated flow $M_2^{\mathsf{S}} \neq \perp$, he sets $K_{\mathsf{C}} := K_{\mathsf{S}}$.

Indistinguishability argument: We only change the simulation in case there is no active corruption. The behaviour of honest parties is the same in both games, so the only way for an adversary to distinguish the two games is by noticing the difference in the creation of protocol values $A, M_1^{\mathsf{C}}, M_2^{\mathsf{S}}, K_{\mathsf{C}}$ and K_{S}. A is indistinguishable since we only removed the simulator's knowledge of a such that $A = g^a$. The other values are outputs of the random oracles H_2, H_3 and H_4 in G_2 while they are uniformly random values sampled from the same domain in G_3. Thus, in order to distinguish them, the adversary has to compute the value $T_{\mathsf{C}} = \mathsf{cdh}_{\mathsf{g}}(Ag^{ux}, B - kg^x)$ and query one of the random oracles on the correct input, i.e. H_2 on (A, B, T_{C}), H_3 on $(A, M_1^{\mathsf{C}}, T_{\mathsf{C}})$ or H_4 on T_{C}. Note that in case of mismatching passwords, the adversary can only distinguish the games using M_1^{C} since the other values are set to $\perp$. We show that the adversary can compute the value T_{C} only with negligible probability if the Gap-CDH assumption holds relative to $\mathcal{G}$ even if the adversary knows the parties' secret inputs pw and pw'.

Therefore, we construct an efficient Gap CDH adversary $\mathcal{B}_1$ interacting with $\mathcal{Z}$. Let (g, g^d, g^e) denote a CDH challenge. $\mathcal{B}_1$ embeds the challenge in the game as follows: $\mathcal{B}_1$ samples $x' \xleftarrow{r} \mathbb{Z}_{p-1}$ and sets $(H_1, [s, \mathsf{uid}, pw'], x')$ where pw' is the client's password. Note that $\mathcal{B}_1$ receives both parties' passwords as input. Then, $\mathcal{B}_1$ sets $A := g^d$ and $B := g^e + kg^{x'}$ for $k = H_5(p, g)$. Now we consider the two cases:

1. $pw = pw'$: If the passwords of both parties match, $\mathcal{B}_1$ samples $s \xleftarrow{r} \{0,1\}^\lambda$, $M_1^{\mathsf{C}} \xleftarrow{r} \mathbb{F}_p$, $M_2^{\mathsf{S}} \xleftarrow{r} \mathbb{F}_p$, $K \xleftarrow{r} \{0,1\}^\lambda$ and simulates the protocol with transcript $(s, B, A, M_1^{\mathsf{C}}, M_2^{\mathsf{S}})$ and output (ssid, K) for both parties. Note that in this case it holds that $T_{\mathsf{S}} = T_{\mathsf{C}} = (B - kg^{x'})^{d+x'} = g^{de+ex'}$. Now for every record of the form $(H_2, [A, B, T], *)$, $(H_3, [A, M_1^{\mathsf{S}}, T], *)$ or $(H_4, [T], *)$, $\mathcal{B}_1$ uses the DDH oracle to check whether $\mathsf{ddh}_{\mathsf{g}}(g^d, g^e, T \cdot ((g^e)^{x'})^{-1}) = 1$. If this is the case, that is, if $T = g^{de+ex'}$, then $\mathcal{B}_1$ outputs $T \cdot ((g^e)^{x'})^{-1}$ and wins in the Gap CDH experiment.
2. $pw \neq pw'$: If the passwords of both parties don't match, $\mathcal{B}_1$ samples $s \xleftarrow{r} \{0,1\}^\lambda$, $M_1^{\mathsf{C}} \xleftarrow{r} \mathbb{F}_p$ and simulates the protocol with transcript $(s, B, A, M_1^{\mathsf{C}}, \perp)$ and output $(\mathsf{ssid}, \perp)$ for both parties. Again, in a real execution of the protocol C would compute $T_{\mathsf{C}} = (B - g^{x'})^{d+x'} = g^{de+ex'}$ and it would hold that $M_1^{\mathsf{C}} = H_2(A, B, T_{\mathsf{C}})$. $\mathcal{B}_1$ checks for every record of the form $(H_2, [A, B, T], *)$ whether $\mathsf{ddh}_{\mathsf{g}}(g^d, g^e, T \cdot ((g^e)^{x'})^{-1}) = 1$ and outputs $T \cdot ((g^e)^{x'})^{-1}$ if this is the case for T. Thus, if the adversary has queried $T = g^{de+ex'}$, then $\mathcal{B}_1$ wins in the Gap CDH experiment.

Hence, if an adversary successfully distinguishes the two games, he can win the Gap CDH experiment with non negligible probability. We thus have

$$|\Pr[\mathsf{G}_3] - \Pr[\mathsf{G}_2]| \leq \mathbf{Adv}^{\mathsf{Gap-CDH}}_{\mathcal{B}_1^{\mathcal{P},\mathcal{G}}}(\lambda)$$

Game G_4: Store possible password candidates and impersonation attempts in H_2, limit number of password guesses for client. Whenever the simulator detects an adversarial password guess or impersonation attempt in the input to H_2, the simulator keeps a record of this. These records will be used to test the simulated party's password in a later game. The game aborts if an input (A, B, T) to H_2 corresponds to two password guesses for different passwords. This essentially limits the client to one password guess per login phase in a later game, as he commits to one value T in the protocol.

Changes to the random oracle: While simulating the random oracle H_2, the simulator keeps records for adversarial password guesses and impersonation attempts. Adversarial password guesses are detected by scanning all records $(H_1, [s, \mathsf{uid}, pw], x)$, $(H_5, [p, g], k)$ and $(H_6, [A, B], u)$. If the simulator finds a combination of these records such that $\mathsf{ddh_g}(Ag^{xu}, B - kg^x, T) = 1$, he keeps record $(\mathsf{guess}, \mathsf{ssid}, pw, T, M_1)$ where M_1 is his answer to the H_2 query. If the simulator finds more than one such combination, the simulation aborts. This ensures that every T value corresponds to at most one password guess. Additionally, the simulator keeps records for adversarial impersonation attempt. Therefore, if the server with $\mathsf{file}[\mathsf{uid}] = (s, v)$ is compromised, he scans all records $(H_5, [p, g], k)$ and $(H_6, [A, B], u)$ to find a combination with $\mathsf{ddh_g}(Av^u, B - kv, T) = 1$. If this is the case, the simulator keeps record $(\mathsf{impatt}, \mathsf{ssid}, pw, T, A, M_1)$ where pw is the password used by the server during registration and M_1 is again the simulator's answer to the H_2 query.

Indistinguishability argument: This change is only internal unless an abort happens. The probability of an abort is negligible since in that case it holds that $T = \mathsf{cdh_g}(Ag^{x_1 u}, B - kg^{x_1}) = \mathsf{cdh_g}(Ag^{x_2 u}, B - kg^{x_2})$ for two passwords pw_1, pw_2 with $x_1 = H_1(s, \mathsf{uid}, pw_1)$ and $x_2 = H_1(s, \mathsf{uid}, pw_2)$. This is the case if

$$\begin{aligned}(g^{a+ux_1})^{\mathsf{DL_g}(B-kg^{x_1})} &= (g^{a+ux_2})^{\mathsf{DL_g}(B-kg^{x_2})} \\ \Leftrightarrow \frac{a + ux_1}{a + ux_2} &= \frac{\mathsf{DL_g}(B - kg^{x_2})}{\mathsf{DL_g}(B - kg^{x_1})}\end{aligned}$$

where $\mathsf{DL_g}$ denotes the discrete logarithm to base g. Since x_1 and x_2 are chosen uniformly at random from $\mathbb{Z}_{p-1}$, the probability of this happening is negligible. Thus,

$$|\Pr[\mathsf{G}_4] - \Pr[\mathsf{G}_3]| \leq (l_{H_1})^2 2^{-q}$$

Game G_5: Abort on impersonation attempt of the client. If the client tries to login using a stolen password verifier v for which he has not correctly guessed the password through an offline dictionary attack, the game aborts.

Changes to the simulation: If the simulator receives a non-oracle-generated flow (A, M_1^{C}) in (S3) such that there is a record $(\mathsf{impatt}, \mathsf{ssid}, pw, T, A, M_1^{\mathsf{C}})$ he aborts the simulation.

Indistinguishability argument: Game 5 is indistinguishable from Game 4 unless an abort happens. We show that the probability of an abort happening

is negligible. The abort happens if there is a compromised password file (s, v), and the adversary $\mathcal{A}$ computed $T = \mathsf{cdh}_{\mathsf{g}}(Av^u, B - kv)$ for $u = H_6(A, B)$ and $k = H_5(p, g)$ without querying H_1 on (s, uid, pw) for the correct password pw. The probability of this happening is negligible if the CDH assumption holds in $\mathbb{F}_p$.

To show this, we construct an efficient CDH adversary $\mathcal{B}_2$ interacting with $\mathcal{Z}$. Let (g, g^d, g^e) denote a CDH challenge. $\mathcal{B}_2$ embeds the challenge in the game by setting $v = g^d$ and $B = g^e + kg^d$. Note that in this scenario the adversary never queries H_1 on the correct password, so the simulation is indistinguishable from G_5 even though the reduction does not know d. Assume that there is an adversary who can compute $T_1 = \mathsf{cdh}_{\mathsf{g}}(Av^{u_1}, B - kv)$ for $u_1 = H_6(A, B)$. By the Forking Lemma [12], the adversary on the same random tape can also compute $T_2 = \mathsf{cdh}_{\mathsf{g}}(Av^{u_2}, B - kv)$ for different random oracle output $u_2 = H_6(A, B)$. $\mathcal{B}_2$ can then combine both intermediary session keys to extract a CDH tuple as $(T_1/T_2)^{1/(u_1-u_2)}$ and win the CDH challenge. If $\mathcal{B}_2$ chooses one of the queries to H_2 at random to extract T_i in each case, then $\mathcal{B}_2$ wins the CDH challenge with non-negligible probability. Thus, the probability of an abort happening is negligible under the CDH assumption.

$$|\Pr[\mathsf{G}_5] - \Pr[\mathsf{G}_4]| \leq l_{H_2}^2 \cdot \mathbf{Adv}^{\mathsf{CDH}}_{\mathcal{B}_2^{\Phi_{\mathcal{G}}}}(\lambda)$$

Game G_6: Limit number of password guesses for server. In this game, the simulator aborts if there is more than one password guess from the adversary, which limits the server to one password guess per login phase.

Changes to the random oracle: The simulator aborts if there are two records $(\mathsf{guess}, pw_i, T_i, *, *)$ and $(\mathsf{guess}, pw_j, T_j, *, *)$ such that $T_i \neq T_j$ for the same session.

Indistinguishability argument: The change is only noticeable if the simulator aborts. We show that this happens only with negligible probability if the aSDH assumption holds in $\mathbb{F}_p$. Therefore, we construct an efficient aSDH adversary $\mathcal{B}_3$ interacting with $\mathcal{Z}$. Let (g, X, Δ) denote an aSDH challenge. $\mathcal{B}_3$ embeds the challenge in this game as follows: $\mathcal{B}_3$ randomly chooses two out of all H_1 queries made by the adversary. Let $x_i := H(s, \mathsf{uid}, pw_i), x_j := H(s', \mathsf{uid}, pw_j)$ be the simulator's output to these queries (note that $s = s'$ is possible, and pw_i, pw_j are the two password guesses of $\mathcal{A}$). The simulator then sets $k := \Delta/(g^{x_i} - g^{x_j})$ as output to query $H_5(p, g)$ and sets $A := X$ as the honest client's message in the protocol. The simulation aborts if there are two values T_i, T_j (which the adversary captures through its DDH oracle) such that $T_i = \mathsf{cdh}_{\mathsf{g}}(Ag^{x_i u}, B - kg^{x_i})$ and $T_j = \mathsf{cdh}_{\mathsf{g}}(Ag^{x_j u}, B - kg^{x_j})$. Defining $Y := B - kg^{x_i}$, it holds that $T_i = \mathsf{cdh}_{\mathsf{g}}(Xg^{x_i u}, Y)$ and $T_j = \mathsf{cdh}_{\mathsf{g}}(Xg^{x_j u}, Y + \Delta)$, hence the adversary wins the aSDH experiments with output (Y, Z_1, Z_2) where $Z_1 = T_i/\mathsf{cdh}_{\mathsf{g}}(g^{x_i u}, Y)$ and $Z_2 = T_j/\mathsf{cdh}_{\mathsf{g}}(g^{x_j u}, Y + \Delta)$. The probability of this event happening is therefore negligible if the aSDH assumption holds in $\mathbb{F}_p$. Thus it holds that

$$|\Pr[\mathsf{G}_6] - \Pr[\mathsf{G}_5]| \leq (l_{H_1})^2 \mathbf{Adv}^{\mathsf{Gap-aSDH}}_{\mathcal{B}_3^{\Phi_{\mathcal{G}}}}(\mathbb{F}_p)$$

Game G_7: Simulate honest server. In this game, the simulator uses the records stored in game 4 to extract a password from the received messages and test it against the server's password.

Changes to the simulation: In (S1) the simulator picks B as a random element from $\mathbb{F}_p$ and aborts if $B = kv$ for $k := H_5(p, g)$. In this game, the simulator still knows the correct value of v and can trigger the abort. In a later game, the simulator is only able to trigger the abort if the server is compromised in which case he knows v. However, the probability of an abort is negligible. After receiving (A, M_1^{C}) in (S3), the simulator checks if there is a record $(\mathsf{guess}, \mathsf{ssid}, pw, T, A, M_1)$. If this is the case and if the password matches the server's password he sets $M_2^{\mathsf{S}} := H_3(A, M_1^{\mathsf{C}}, T)$ and $K_{\mathsf{S}} := H_4(T)$. In every other case, he sets $(M_2^{\mathsf{S}}, K_{\mathsf{S}}) := (\perp, \perp)$.

Indistinguishability argument: If (A, M_1^{C}) is constructed as per the protocol, the simulator is able to extract the adversary's password and simulate the protocol indistinguishably. If (A, M_1^{C}) is not constructed as per the protocol, then the simulator sets $(M_2^{\mathsf{S}}, K_{\mathsf{S}}) := (\perp, \perp)$ which is also an indistinguishable simulation of the protocol. Recall here that the sid and ssid are prefixed to H_2 inputs, which prevents the attacker from passing off a valid M_2^{S} from a different ssid'. Hence, G_6 and G_7 are indistinguishable unless an abort happens. Since $v = g^x$ is initialized with a random $x \xleftarrow{r} \mathbb{Z}_{p-1}$ chosen by the simulator, the probability that $B = kv$ and therefore an abort in game G_7, is negligible. Thus, it holds that

$$|\Pr[\mathsf{G}_7] - \Pr[\mathsf{G}_6]| \leq \frac{1}{p}$$

Game G_8: Simulate honest client. In this game, the simulator makes use of the records stored in game G_4 to simulate the client. By using these records he can extract a password guess or detect an impersonation attempt from the messages received. This password check is actually performed in a H_2 query. The simulator can test the extracted password against the client's password and adjust the output of the H_2 query accordingly.

Changes to the simulation: In (C2), the simulator samples random values A, M_1^{C} also in the case that the message received in (C2) is non-oracle generated. In (C4), upon receiving a non-oracle-generated flow M_2^{S} the simulator checks if there is a record $(H_3, [A, M_1^{\mathsf{C}}, T], M_2^{\mathsf{S}})$ and if $H_2(A, B, T) = M_1^{\mathsf{C}}$. If this is the case, the simulator sets $K_{\mathsf{C}} := H_4(T)$, and $K_{\mathsf{C}} := \perp$ otherwise.

Changes to the random oracle: If there is a query to H_2 where the simulator would store a record $(\mathsf{guess}, \mathsf{ssid}, pw, T, *)$, the simulator first checks if he needs to "backpatch" the output of the random oracle query to match the client's randomly chosen M_1^{C}. If the client's password matches the password pw extracted from the query he programs the output to M_1^{C}, otherwise he picks a different M_1^{C} uniformly at random. The same backpatching is also performed for records $(\mathsf{impatt}, \mathsf{ssid}, pw, T, *, *)$.

Indistinguishability argument: The handling of H_2 queries ensures that the games are indistinguishable unless the adversary manages to guess the output of one of the hash functions without querying the random oracle. The probability of this happening is negligible. Furthermore, recall that the sid and ssid are prefixed

to H_3 inputs, which prevents the attacker from passing off a valid M_2^{S} from a different ssid'. Thus, it holds that

$$|\Pr[\mathsf{G}_8] - \Pr[\mathsf{G}_7]| \leq \mathsf{negl}(\lambda)$$

Game G_9: Change password file creation and StealPwdFile interaction.

Changes to the simulation: This change is identical to the change made in G_5 of the proof for Theorem 1.

$$|\Pr[\mathsf{G}_9] - \Pr[\mathsf{G}_8]| \leq l_{\mathsf{ssid}} \cdot 2^{-q}$$

Game G_{10}: Introduce $\mathcal{F}$, password check is handled by $\mathcal{F}$, simulator does not receive private inputs. In this game we add an ITI $\mathcal{F}$ external to the simulator. $\mathcal{F}$ has all interfaces of $\mathcal{F}_{\mathsf{aPAKE}}$ and manages all password checks in the protocol. Furthermore, the ideal functionality provides the session keys to the parties. We thus remove the simulator's access to the parties' private inputs.

Changes to the simulation: We introduce functionality $\mathcal{F}$ who handles all password checks via TestPwd queries for passwords that were extracted from messages and NewKey queries in the simulation of honest parties with all flows oracle-generated. An Impersonate query is used to check the password for records impatt. The simulator sends out the keys to the parties via NewKey queries and does not receive the private inputs of the honest parties. The records impatt are stored without the passwords.

Indistinguishability argument: The changes are only internal, and thus

$$\Pr[\mathsf{G}_9] = \Pr[\mathsf{G}_{10}] = \Pr[\mathsf{Ideal}(\mathcal{F}_{\mathsf{aPAKE}}, \mathsf{SIM}^{\Phi_{\mathcal{G}}})]$$

G_{10} is identical to the ideal execution which concludes the proof.