Analysis of J-PAKE - a PAKE-Function

Nicolas Patrick Schweinhuber
Technische Universität München
Munich, Germany

***Abstract*—In our modern world the data traffic via the internet increases steadily. To protect transactions and sensitive user data from attackers a secure transmission of the data is of great importance. Passwords help to keep data secure but choosing and remembering a secure password is often complicated for human users. In this paper we will take a close look at "J-PAKE" which is one "Password Authenticated Key Agreement" protocol which operates securely with simple passwords. We will look at the general idea of PAKE, how such protocols can be classified and which basic concepts we need to understand the algorithm itself. The next section covers the goals of J-PAKE concerning active and passive attacks on the communication. To analyze the fulfillment of these goals we look at the algorithm and how the session keys for a secure communication are calculated and provide a simple example. As zero-knowledge proofs are a characteristic of J-PAKE we will analyze why the use is necessary in this protocol. Our extensive analysis also contains a look at the performance compared to well known PAKE functions. After looking at all theoretical parts we take a look at a real world implementation for the "sync" feature in the Firefox browser and why it got removed.**

***Index Terms*—J-PAKE, PAKE-function, PAKE**

I. Introduction

In our everyday life we encounter many different online services and websites where we have multiple different accounts. Most of them are linked to our email address and require a password. A problem well known to many users is choosing and remembering a secure password which is hard to guess and hard get hacked. Because the ordinary user isn't aware of all requirements for a secure password or won't change them from time to time this poses a risk to the users private data. An approach that allows users to use simple passwords and remain a high standard of security is password authenticated key agreement, short PAKE. Among many different protocols for PAKE we look at password authenticated key agreement by juggling called "J-PAKE". Before we take a look at the actual algorithm we introduce some basic concepts the algorithm is based on. Especially zero-knowledge proofs are of great importance as this is a core element of J-PAKE. Further topics discussed are the decisional Diffie-Hellman assumption and cyclic groups with a special focus on "Schnorr Groups". Nowadays it's nearly impossible to talk about cryptography without crossing the field of elliptic curves. Even if the original J-PAKE idea isn't based on it, it can be used for a modern approach of it. That's why we will cover it shortly to mention some important advantages of using ECC. The purpose of J-PAKE is to prevent active and passive attacks against an online end-to-end communication either between two clients or between client and server. The specific goals are to guarantee resistance against "off-line dictionary attacks" and "on-line dictionary attacks". Other aspects are "forward secrecy" to produce secure session keys and "known-key security" to keep other sessions safe from a disclosed session key. PAKE protocols can be classified either as augmented or as a balanced protocol. J-PAKE belongs to the balanced category which means both party members use the same password to authenticate a shared key. Furthermore PAKE functions can be divided in five different design classes enumerated form "C1" to "C5". J-PAKE is a function implementing password authenticated key agreement and belongs to the C4 design class. The class "C4" is characterized by using zero-knowledge-proofs to check that every party member follows the protocols specifications. J-PAKE uses a protocol consisting of two steps where the first step has two rounds that are mandatory for the key establishment and the second step is an optional key confirmation. In the first round the individually chosen public keys of the participants are exchanged and protected by zero-knowledge proofs. These information will be used by each party to compute a one-time session key. In round two the members exchange their results from round one again secured by ZKPs. At the end of the second round both parties will have the same session key which will be used to encrypt the rest of the communication between the two parties during the session. How this works will be shown in the next section with some simple numbers. To complete the analysis of J-PAKE we will compare it to two well known PAKE protocols called EKE and SPEKE. We will look at their performance concerning the computational costs and the potential to optimize implementations of the protocols. At this point we covered all the theory about J-PAKE and in the end we can use our knowledge to look at an example of an implementation that was used in the real world. This implementation could be found in the "sync" feature of the Firefox browser which was used to synchronize the users data between different devices using a pairing code as a shared key. We will discuss how it worked and why it got replaced by the developers. At last we will look back at everything discussed in this paper and draw conclusion about how people can benefit from this analysis.

src?

II. What is PAKE?

A. Purpose of PAKE

Like mentioned in the introduction in the real world the vast majority of users are no experts in terms of cyber-security. So they can't be expected to have a focus on creating ordinary secure passwords for every service they use on the internet. Because short and simple passwords have a low entropy they are vulnerable to different sorts of guessing attacks. This is where password authenticated key exchange comes into play. Protocols of this field are designed to be secure against many of these attacks even with small passwords. The attacker will only be limited to online guessing attacks. [1] [5]

but can a service not limit attempts?

B. Difference between PAKE and PKI

Since a paper of Feng Hao and Peter Ryan who are of great importance in the field of PAKE runs by the name of "J-PAKE: Authenticated Key Exchange Without PKI" [5] looking at the difference of these concepts is worth a look. PKI which is the abbreviation for "Public Key Infrastructure" is a widespread system which is based on a hierarchy of trust among different institutions which issue certificates that guarantee trust in the integrity of the connection to sites using these certificates. Both approaches PKI as well as PAKE have in common that they are used in use cases where simple passwords are not suitable. The approach of PAKE is to be independent from the hierarchy of trust where a certificate authority (CA) or registration authority (RA) is needed. PAKE instead relies on establishing a connection between two parties where both parties have a shared secret which is used to negotiate an encrypted communication. The advantages and disadvantages depend on the use case. So PAKE grants us more independence compared to PKI but demands more computational costs for the client to generate the keys. PAKE protocols are often based on the computation of exponentiations on both sides of the connection which is more complex then issuing a certificate and working with public keys in PKI.[5]

III. J-PAKE in the context of all PAKE-functions

A. J-PAKE in the History of PAKE Research

This section is about the history of all PAKE functions. It started in 1992 with PEKE basically implementing the Diffie-Hellman key exchange. In the time period from 1992-2008 there was great interest of the industry in this new research field and also of IEEE to form a standardization for it. Many new concepts and functions got developed in this time span to test the theoretical approach of PAKE in practice e.g. SPEKE, PAK and AMP. In this time and ongoing a lot of revisions and modifications were needed because a majority of the functions were suffering from security issues.[4] In a ten year time span from 2008-2018 the focus was on standardizing PAKE. Since the PAKE problem wasn't solved at the begin of this time more research was done and new protocols were invented and added to the standard. For this paper this is especially interesting because J-PAKE was added to the ISO/IEC 11770-4. In addition to that PAKE made its way from the theory to the practice by being implemented in real world application. This was also the case for J-PAKE and it got adopted by the "Thread Group" in 2015. After 2018 a third period of research began that is going on until the present day. In this time span PAKE got integrated into TLS 1.3 because the pre-shared key (PSK) serves as an ideal use case to implement PAKE to establish a connection with a pre-shared key.[4]

what is the pake problem?

B. augmented and balanced PAKE

Two big groups PAKE-functions can be separated in are augmented and balanced functions. Augmented functions are used in client/server scenarios and ~~they ensure that a client won't gain knowledge of the servers salt (a random sequence of chars added to the input to raise the entropy).~~ On the other side it's ensured that the server doesn't know the clients password. Balanced functions to which J-PAKE belongs can be used in client/server as well as client/client scenarios. In these functions both parties use the same password to authenticate a shared key. [4]

details?

C. overview of design classes

Independent from the classification of augmented or balanced, PAKE-functions can be assigned in different design classes. These classes are enumerated from C1 to C5 and can be distinguished by their use and interpretation of the password.[4]

1) C1: password as encryption key
2) C2: password to derive a protocol group generator
3) C3: multiple generators with unknown discrete logarithm relationships a trusted setup relies on
4) C4: Two-party secure computation problem on an equality function. That parties follow specifications is checked via zero-knowledge proof (ZKP)
5) C5: password used to derive g^w in a DH key exchange

In our case J-Pake ist especially interesting because according to the recently distributed paper from 2021 of Hao and Oorschot [4], it's the only function enlisted as a class 4 function. In the following we will have a look at the important components of the algorithm to understand the function as a whole.

IV. Components of the algorithm

A. Discrete Logarithm Problem

"Discrete logarithms are logarithms defined with regard to multiplicative cyclic groups. If G is a multiplicative cyclic group and g is a generator of G, then from the definition of cyclic groups, we know every element h in G can be written as gx for some x. The discrete logarithm to the base g of h in the group G is defined to be x." [8] To make a group safe, it's important to have prime number "p" of a certain length (at least 1024 Bit) so the discrete log can't be computed easily by any algorithms. [8] This problem forms the base for the following aspects "DDH", "Zero-Knowledge Proof" and the "Schnorr Signature".

B. Decisional Diffie-Hellman Problem (DDH)

The purpose of DDH is creating a possibility for a formal proof of a systems security. Building up on the "Discrete Logarithm Problem" of the previous section the DDH is looking for a multiplicative group for which the "Decisional Diffie-Hellman Assumption" is valid.[2] Again we're looking at a cyclic, finite group G which is a subgroup of $Z_p^\times$ where p is prime. G has order q and generator g and for each element of G exists an exponent $x \in Z_p$, so $h = g^x$. To see if the assumption holds we need three numbers $x, y, z \in Z_q$ to form g^x, g^y and g^z. In the DDH x and y are chosen randomly and the probability that z is either $x \cdot y$ or it is chosen randomly must be $1/2$ for both possibilities. If there exists no efficient algorithm for G granting more than a 50% success rate the DDH assumption is valid for this group.[2]

C. Zero-Knowledge Proof

Zero-knowledge proofs are encryption schemes. The basic idea behind them is that one party in a communication scenario can prove to another one that they know something but is able to prove their knowledge without actually telling the answer. Every scenario can be seen as an interaction between two party-members we label as prover and verifier. In the field of zero-knowledge proofs we differentiate between interactive and non-interactive proofs. As the name suggests for the first one a specific interaction between the two individuals is mandatory. In this scenario the prover is revealing his hidden information only to the verifier and no other "spectator". This is on one hand-side an advantage when it comes to security/privacy. On the other side this functionality comes with more complexity and makes it harder to confirm the provers knowledge to multiple instances. This is where we get to the second variant, the non-interactive zero-knowledge proof. This type lacks the additional privacy of the first one but convinces with a faster verification for different observers because there is no interaction between every single instance. In the following explanation of the J-PAKE function we deal with the second type, the non-interactive zero-knowledge proofs. [13, 14]

D. Schnorr Signature

A digital signature is a mathematically created value used for the verification of a digital message. They are commonly used to transmit sensitive data via insecure connections.[wiki] Among many procedures to create digital signatures the Schnorr Signature is quite simple and efficient (allows to combine multiple signatures into one) [9]. The base the algorithm is formed on is choosing a Group g which is hard in regards of the discrete log problem. For this procedure a certain type of prime-group is used, a so called Schnorr Group. [12] and then ?

E. Elliptic Curve Cryptography

We already had a look at the Diffie-Hellman problem which is one of the fundamental problems many key encryptions are based on. A modern approach gaining more and more popularity in the field of cryptography is "elliptic curve cryptography", short ECC [10]. This new approach is based on the elliptic curve discrete logarithm problem which is mathematically very complex. Similar to other discrete logarithm problems we have a trap-door function which means doing a procedure is quite easy but undoing the same is very hard. In the case of ECC we imagine the graph of an elliptic curve and on this graph we perform operations to move from a starting point on the graph over multiple other points to an end point. This can be done easily following pre-defined rules but only knowing the start and end point it is still hard to reverse the procedure. It is such an interesting topic in our case because these problems are much harder to crack that the Diffie-Hellman approach and on the other hand they allow the use of smaller numbers which allows more efficiency on mobile devices. These aspects are advantages of ECC making it an important topic for the future and it's interesting to see that it is applicable to protocols like J-PAKE that were initially based on older methods.[10]

trapdoor ?

F. Attackers in cyber security

As J-PAKE is used to ensure security for communication on the internet it's necessary for us to look at the risks and possible attacks that can be performed. In general attacks can be split into two groups, the active attackers and the passive attackers we will have a closer look on in the following paragraphs. We will start off by looking at passive attacks because the explanation for the active ones can be built up on this. The design goals and purpose of J-PAKE in regards of attacks will be covered in the next section.

1) passive attacker: The main intention behind a passive attack is gaining access to the victims system. Once in the system the attacker will analyze it to find vulnerabilities to get further access and knowledge about sensitive data of the user. These activities can be difficult to detect.[11]

2) active attacker: The purpose of an active attacker instead of a passive one is "actively" changing or modify data to impersonate themselves as another identity which grants them access to restricted areas. This access can be gained by exploiting the weaknesses found during a passive attack. However an active attack can be much more dangerous compared to a passive one these activities are more likely to be detected.[11]

V. The J-PAKE Algorithm

Now that we have covered the basic components concerning the J-PAKE function we take a closer look at the function itself. At first its main purpose as intended by the creators. Then how the different parts interact with each other as a whole. At first there will be a description of the theory behind the function followed by an example with simple numbers [7] to deepen the understanding.

A. Goals of J-PAKE

To ensure the security of our procedure we have to cover different aspects and variants of attackers and how they could possibly harm our communication.

1) offline dictionary attack resistance: A passive attacker tries to gather information about a password. With this information he can try later on to authenticate himself as a valid communication partner. The goal is to prevent the leak of such information.[6]
2) Forward secrecy: If an attacker tracked encrypted messages and later somehow gets to know the disclosed password the session keys still should be secure.[6]
3) Known-key security: If the key of a session is disclosed the other sessions shouldn't be affected by that and remain secure.[6]
4) online dictionary attack resistance: Online dictionary attacks are based on the attackers guesses of a password. With these guesses he tries to enter the communication. The goal is to prevent a large amount of easy executable guesses.[6]

Now that we know about these goals and different attack variations we will at first take a look at the algorithm of J-PAKE in general. After gaining detailed knowledge of its functionality we take a look at if and how these goals can be fulfilled.

VI. Theory of J-PAKE

The algorithm is designed as a two-step exchange. The first step is the establishment of a session key. This process is running in two rounds. After the second round both participants of the connection will have the same session key. The second step is optional in the J-PAKE protocol and is about key confirmation. The purpose of this step is to reassure that after the first step is complete the session key of the members really are the same. As commonly used in descriptions of encryption processes we will name the two individuals Alice ("A") and Bob ("B").

At first we need a group G which is chosen as a subgroup of $\mathbb{Z}_q^*$ and g which is a generator in G where both parties agree on the combination of (G, g). Most commonly a Schnorr Group is used for this process. We have the group G with prime order q, it's generator g formed as a subgroup of $\mathbb{Z}_q^*$ which is a multiplicative group of integers modulo the prime p. One more component is needed, the shared secret s. Defined by the use case that we have low entropy passwords we choose s from an interval of $(1 \leq s \leq q-1)$. [6, 7]

Why?

A. ZKP for J-PAKE

Zero-knowledge proof form a core element of the J-PAKE algorithm and are inevitable when it comes to guaranteeing security against impersonation attacks. How they work in detail and what they are exactly needed for will be explained in the following sections.

B. How do ZKPs work in J-PAKE?

The concept of a ZKP is already explained in an earlier section. Here we go into detail how we perform a proof in J-PAKE. We want to show that if one party sends $X = g^X$ the other party knows the exponent. To construct such a proof the sender computes a value h by using a secure hash function H.[7]

what is the X?

1) $h = H(g, V, X, ID)$
2) g is the generator
3) we need a random v picked from Z_q to get $V = g^v$
4) $X = g^x$
5) ID is used to uniquely identify the members

order of steps??

The sender sends X, ID, V and $r = v - x \cdot h$

Now it's the receivers turn to compute

$$h' = H(g, V, X, ID)$$

With this we get $V' = g^r \cdot X^{h'}$ and so we can check if $V = V'$.[7]

v confusing expl. no intuition provided

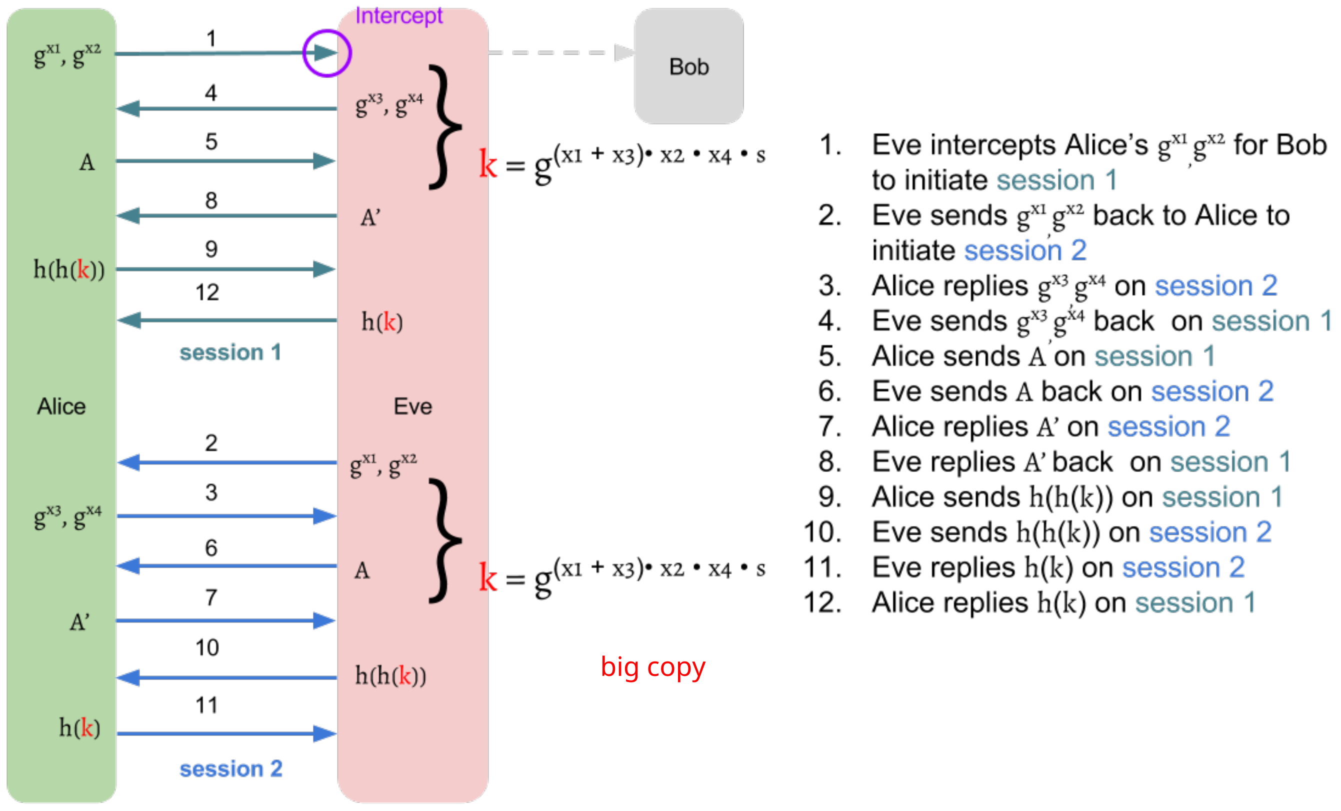
C. Why do we need ZKPs in J-PAKE?

One question coming up about this topic is if it would be sufficient to use a hash function to provide more security for the exchanged information. The answer in this case is no. If we would use a hash function both parties would have to have the same secret to verify the received value. If this isn't the case the result can't be compared so a hash isn't really suitable for our use case in J-PAKE.[7] The reason behind the usage of zero-knowledge proofs in J-PAKE is the prevention of impersonation-attacks. This attack would allow an attacker "Eve" (eavesdropper) to make it seem like he is the one the initiator of the communication wants to communicate with.[7] Figure 1 shows us what an attacker can do if we don't secure our messages with zero knowledge proofs. What we see here is a so called impersonation-attack. The functionality can be derived from the name meaning that the attacker impersonates himself as one of the communication partners to establish a connection on his own. The proofs show us that the other party member has knowledge that authorizes him to be a valid communication partner. In Figure 1 this is not checked, so what the attacker does is establishing two connections to Alice. In the first session he gathers values sent by Alice and sends them in the second session to mirror the behavior of Alice to get the values she computes to send them back to establish a connection with correctly calculated values. Since the mechanisms behind the computation produce the same output for the same input the values are valid and will be accepted by Alice. To prevent this the use of the zero-knowledge proofs is required.[7]

bit redundant text blown up previsously simple explanations

D. Step 1: Key Establishment

1) Round 1: Alice and Bob send each other messages each containing four elements. Because the elements can be chosen independently the messages can be sent simultaneously. The range for x_1 and x_3 is different from x_2 and x_4. This is necessary because x_2, x_4 and s can't be 0. If one of these



1. Eve intercepts Alice's g^{x1}, g^{x2} for Bob to initiate session 1
2. Eve sends g^{x1}, g^{x2} back to Alice to initiate session 2
3. Alice replies g^{x3}, g^{x4} on session 2
4. Eve sends g^{x3}, g^{x4} back on session 1
5. Alice sends A on session 1
6. Eve sends A back on session 2
7. Alice replies A' on session 2
8. Eve replies A' back on session 1
9. Alice sends h(h(k)) on session 1
10. Eve sends h(h(k)) on session 2
11. Eve replies h(k) on session 2
12. Alice replies h(k) on session 1

big copy

Figure 1. Impersonation Attack [7]

components would be 0 it would result in a security problem because K can be forced to be $K = 1$ without knowledge of other components. For every calculation in the following steps it's necessary to keep in mind that every result must be computed with mod p so our values stay within our selected group. Alice starts by selecting $x_1 \in [0, q-1]$ and $x_2 \in [1, q-1]$. The four elements Alice sends are g^{x_1} and a zero-knowledge-proof for x_1 as well as g^{x_2} and a zero-knowledge-proof for x_2. Bob selects $x_3 \in [0, q-1]$ and $x_4 \in [1, q-1]$. Then he constructs his message containing g^{x_3} with zero-knowledge-proof for x_3 and g^{x_4} with zero-knowledge proof for x_4. If one party member receives the message from the other one it's necessary to verify the zero-knowledge-proofs. In addition to that, Alice checks if the received $g^{x_4} \neq 1$ and Bob does the same for g^{x_2}. This is necessary to avoid that an attacker can easily manipulate values. If one of these exponentiations is equal to one the base of the exponentiation is one and will always be one, no matter what the exponent is. This is similar to our previous check that x_2, x_4 and s can't be 0.[7]

2) Round 2: With the elements received in round 1 Alice send

$$A = (g^{x_1} \cdot g^{x_3} \cdot g^{x_4})^{(x_2 \cdot s)}$$

and a ZKP for $x2 \cdot s$. Bob constructs the variable

$$B = (g^{x_1} \cdot g^{x_2} \cdot g^{x_3})^{(x_4 \cdot s)}$$

and a ZKP for $x_4 \cdot s$. Now it's time to compute K, a value from which a session key can be derived.[7]

Alice computes

$$K = (B/g^{(x_2 \cdot x_4 \cdot s)})^{x_2}$$

$$B = (g^{x_1} \cdot g^{x_2} \cdot g^{x_3})^{(x_4 \cdot s)}$$

$$= g^{(x_1 \cdot x_4 \cdot s)} \cdot g^{(x_2 \cdot x_4 \cdot s)} \cdot g^{(x_3 \cdot x_4 \cdot s)}$$

spacing?

so

$$B/g^{(x_2 \cdot x_4 \cdot s)}$$

$$= g^{(x_1 \cdot x_4 \cdot s)} \cdot g^{(x_3 \cdot x_4 \cdot s)}$$

$$= g^{(x_1 + x_3) \cdot x_4 \cdot s}$$

$$K = (g^{(x_1 + x_3) \cdot x_4 \cdot s})^{x_2}$$

$$= g^{(x_1 + x_3) \cdot x_2 \cdot x_4 \cdot s}$$

Bob computes

$$K = (A/g^{(x_2 \cdot x_4 \cdot s)})^{x_4}$$

$$A = (g^{x_1} \cdot g^{x_3} \cdot g^{x_4})^{(x_2 \cdot s)}$$

$$= g^{(x_1 \cdot x_2 \cdot s)} \cdot g^{(x_3 \cdot x_2 \cdot s)} \cdot g^{(x_4 \cdot x_2 \cdot s)}$$

so

$$A/g^{(x_2 \cdot x_4 \cdot s)}$$

intuition ? explanation?

$$= g^{(x_1 \cdot x_2 \cdot s)} \cdot g^{(x_3 \cdot x_2 \cdot s)}$$
$$= g^{((x_1 + x_3) \cdot x_2 \cdot s)}$$
$$K = (g^{(x1+x3) \cdot x2 \cdot s})^{x_4}$$
$$= g^{(x_1+x_3) \cdot x_2 \cdot x_4 \cdot s}$$

So now both parties Alice and Bob possess the same K. The last step to get a session key is for Alice and Bob to use a cryptographic hash function H to compute $k = H(K)$.[7]

E. Step 2: Key Confirmation

This step is optional in J-PAKE and is designed to guarantee extra security. So the parties authenticate each others session key to check if they are equivalent. One way this can be achieved is by a ”Hash-Key Authentication”. For this process we need a hash function H. Alice sends

$$C^A = H(H(k))$$

Bob puts his own k into $H(H(k))$ and checks if it equals the C^A he received. Bob responds with

$$C^B = H(k)$$

Now Alice can check if $H(C^B)$ equals the C^A she created initially. There are more procedures that can be used to process this check e.g. a ”Challenge-Response Authentication”.[7]

VII. J-PAKE Numeric Example

We start by creating our group G, more specifically we use a Schnorr Group.[7] So let’s take a look at the formula

$$p = q \cdot r + 1$$

which is used to generate such a group. If we generate our variables $q = 11$ and $r = 2$ we can calculate p.

$$p = 11 \cdot 2 + 1 = 23$$

So our generated group is $\mathbb{Z}_{23}^*$. From this group we need to choose a subgroup generated with our generator g.[7] e.g.

$$g = 9$$

Every value x, x element $\mathbb{N}$ can be used on the generator function g^x. The result will be calculated with modulo p (23). After this step we get the group

$$G = 1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18$$

which contains every modulo-class for our given values.[7] The shared secret is:

$$s = 7$$

Now the participants choose their values according to the protocols specifications(ref dazu). Alice selects:

$$x_1 = 9$$
$$x_2 = 3$$

Bob selects:

$$x_3 = 5$$

ŝtraight taken from source

$$x_4 = 6$$

Round 1:
Alice sends

$$g^{x_1} = 2 \text{ and } g^{x_2} = 16$$

Bob sends

$$g^{x_3} = 8 \text{ and } g^{x_4} = 3$$

Now that Alice and Bob both know the values x_1, x_2, x_3, x_4 we can compute the values A and B with the formula shown in section VI.

$$A = (g^{x_1} \cdot g^{x_3} \cdot g^{x_4})^{x_2 \cdot s}(mod p)$$
$$A = (2 \cdot 8 \cdot 3)^{3 \cdot 7}(mod 23)$$
$$A = 12$$

$$B = (g^{x_1} \cdot g^{x_2} \cdot g^{x_3})^{x_4 \cdot s}(mod p)$$
$$B = (2 \cdot 16 \cdot 8)^{6 \cdot 7}(mod 23)$$
$$B = 18$$

Round 2:
Alice send A with a ZKP for $x_2 \cdot s$ to Bob. Bob sends B to Alice with a ZKP for $x_4 \cdot s$
Alice computes the session key K with:

$$J_{Alice} = B \cdot g^{-(x_2 \cdot x_4 \cdot s)}$$
$$J_{Alice} = 18 \cdot 9^{-(3 \cdot 6 \cdot 7)} = 8$$
$$K = (J_{Alice})^{x_2}$$
$$K = 8^3 = 6$$

Bob computes K with:

$$J_{Bob} = A \cdot g^{-(x_2 \cdot x_4 \cdot s)}$$
$$J_{Bob} = 12 \cdot 9^{-(3 \cdot 6 \cdot 7)} = 13$$
$$K = (J_{Bob})^{x_4}$$
$$K = 13^6 = 6$$

Here we can see that in the end K is the same on both sides. At this point the optional key confirmation step explained in the theory section is possible.

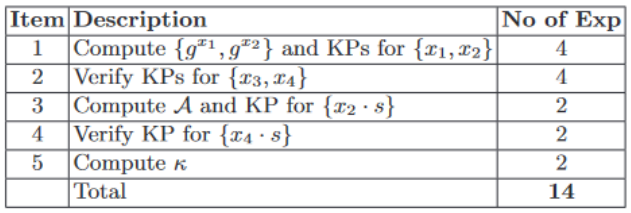
A. Fulfillment of the Goals

Now that we went through the algorithm in detail we can come back to our previously discussed goals. So we can answer the questions whether the goals are fulfilled and how this is done. In general one aspect securing the communication is the shared secret. As discussed previously we use zero-knowledge proofs to authenticate that the party we communicate with knows certain information. First we look at an offline dictionary

attack. The problem here is that an attacker tries to gather information from eavesdropping the communication about the shared secret. We prevent the passive attacker from listening to information about the shared secret simply by not transmitting it in the connection establishment process.[5]
The second goal we look at is the forward secrecy. So we want to produce session keys that remain safe even if the secret is disclosed later on.[5] We assume that the attacker knows the password but also needs knowledge about the random values x_1 and x_2 issued by Alice. In this case to build the connection the attacker issues x_3 and x_4 by himself and are randomly chosen as well. Our assumption that the forward secrecy holds is based on the extremely low probability of computing x_1 and x_2 correctly.[7]
Next let us see how the known-session security is implemented. If an attacker managed to successfully enter one session we don't want other sessions to be affected by the gained knowledge from this compromised session. This is solved by the random generation of the key K. Like we have seen in the sections above the process of creating K relies on both parties choosing the ephemeral values x_1, x_2 from Alice and x_3, x_4 from Bob. So the creation of K is random in each session and not related to an other established session. This guarantees that the interception of one sessions information will affect the security of any other session.[5]
The last goal mentioned in section five is the online dictionary attack. By performing this attack an active attacker tries to guess our secret s and wants to use it to establish a connection. In this case the success of the attack simply depends if the guessed s' of the attacker is equal to the actual secret s or not. The best case besides guessing the correct secret is gaining knowledge that s' is incorrect. The algorithm itself doesn't prevent a malicious entity from performing these guesses but in a real world implementation we can simply limit the tries and detect such behavior to perform countermeasures.[5]

B. J-PAKE in Comparison with other PAKE-functions

Among the many existing PAKE functions we need to find some that are suited for a good comparison. Since J-PAKE belongs to the category of the balanced functions we take a look at others belonging to this category. As shown in the SoK about PAKE of Feng Hao and Paul C. van Oorschot J-PAKE is the only one belonging to design-class 4, so we can't get more specific in this direction. We will compare J-PAKE to EKE and SPEKE.[5] Both protocols are well known, simple and efficient, so they form a good alternative that we can use to compare with. The first difference we notice is that EKE as well as SPEKE are executed in one round but J-PAKE requires two rounds. The next aspect we look at is the amount of exponentiations that need to be performed by each of the users. For both EKE and SPEKE it's two exponentiations per user. Using J-PAKE we need 14 per user which can be derived from Figure 2.



Item	Description	No of Exp
1	Compute $\{g^{x_1}, g^{x_2}\}$ and KPs for $\{x_1, x_2\}$	4
2	Verify KPs for $\{x_3, x_4\}$	4
3	Compute $\mathcal{A}$ and KP for $\{x_2 \cdot s\}$	2
4	Verify KP for $\{x_4 \cdot s\}$	2
5	Compute κ	2
	Total	**14**

Figure 2. exponentiations needed for one member in J-PAKE [5]

Based on these differences it looks like J-PAKE can't compete with the performance of the other protocols which will change by looking at the length of the exponents that are computed in the functions. To grant strong security EKE and SPEKE need big exponents so they use a 1024-bit p and 160-bit q. J-PAKE on the other hand has more steps and grants additional security through the use of zero-knowledge proofs so it can use smaller exponents. The computational cost of an exponentiation rises with the size of the exponent. By taking a closer look at the size we see that one exponentiation in EKE/SPEKE has about the same computational costs as 6-7 exponentiations in J-PAKE. By summing this up we notice that two exponentiations in EKE/SPEKE are of nearly the same costs as the 14 needed for J-PAKE.[5] An advantage of J-PAKE over the other protocols is that these numerous smaller calculations provide more potential for optimization when implemented as code. Many of the computed variables in J-PAKE are repeated with nearly the same input. This is no option for the small amount of big calculations in EKE and SPEKE.[5]

this comparison is pretty much
straight from the paper

C. Real World Implementation of J-PAKE

After looking at all the theoretical aspects of J-PAKE, now it's time to look at implementations where this concept was used in the real world and in applications we might use ourselves.

D. Mozilla Firefox "Sync Feature"

One of the most popular applications where J-PAKE was used is the web browser Firefox from Mozilla. The PAKE function was implemented for a feature called "Sync 1.0"[3]. Its purpose is to make it possible for users to share the user specific information of their browser among different devices e.g. the browser history or saved passwords. The basic idea of the implementation was that a secret key is generated which will be embedded in a so called "pairing key" that was displayed on the users device. This "pairing key" could be entered on a new device to set everything up. By using this key the encryption and decryption is only possible for the user and not even the server knows about it. This is a good use case for J-PAKE not only because it keeps the user information stored on the account as private as possible but also takes away responsibility from the user to create or remember a password. This limits the risks on the human side which has been mentioned in the introduction. This implementation was used for "Sync" from 2010-2014. In April of 2014 it got replaced because even if the idea behind it sounds reasonable some problems appeared and Mozilla wanted to change their approach for this feature. The problems were actually not on a technical level but there was a need for a more suitable solutions for human users. The main problem was that the data of a user owning only one device could get entirely lost if the device got lost or damaged. If Firefox couldn't get accessed on that device there was no way to get to the key to restore the data on a new device. A few more problems were that for the pairing process it was necessary for both devices to be physically close to each other. On top the "pairing code" got mistaken for a password users were meant to remember. To avoid these usability issues Mozilla switched to an account system as we know it from many services. A user account is identified and protected by an email address and a password chosen by the user. This makes it possible to gain access to the account on every device, where ever you are and data can easily be restored. On the other hand it shifts responsibility back to the user to choose and remember a strong password which can lead to other problems. In the end this is a trade in decision making which Mozilla chose in this case.[3]

VIII. Conclusion

Now that we took a close look at all aspects of J-PAKE we can come to a conclusion for whom this topic and this paper is useful. J-PAKE is an interesting PAKE protocol to look at for many different aspects. First if you are interested in PAKE in general it's worth to look at as it is the only member of class 4. If you have an interest in cryptography and security in general it also covers various parts. Especially the main feature of J-PAKE being the use of zero-knowledge proofs as well as the basics e.g. being built up on the Diffie-Hellman Problem. It's also compatible with newer approaches like elliptic curve cryptography and on top J-PAKE is in comparison to many other PAKE protocols a protocol with implementations used in the real world when others were just theoretical concepts or their implementations needed many revisions. Even though these implementations were replaced by other approaches like in Firefox or never actively used like in OpenSSL. The prime time of J-PAKE was between 2008 and 2014 but PAKE is still an evolving topic both approaches, the balanced one as well as the augmented ones. The newest members in each category are "CPace" (balanced) from 2019 and "KHAPE" (augmented) from 2021. So research in this field is still ongoing as cryptography is an ever evolving topic in terms of cyber security.[4] Looking at these developments this paper forms a good starting point for the basic understanding of PAKE as well as a stepping stone for further research on newer topics in this field.

References

[1] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. "LNCS 3386 - Password-Based Authenticated Key Exchange in the Three-Party Setting". In: (2005). URL: http://www.di.ens.fr/users/\{mabdalla, fouque,pointche\}.

[2] Dan Boneh. "The Decision Diie-Hellman Problem". In: ().

[3] *Firefox Sync's New Security Model — Mozilla Services*. URL: https://blog.mozilla.org/services/2014/04/30/firefox-syncs-new-security-model/.

[4] Feng Hao and Paul C Van Oorschot. "SoK: Password-Authenticated Key Exchange – Theory, Practice, Standardization and Real-World Lessons; SoK: Password-Authenticated Key Exchange – Theory, Practice, Standardization and Real-World Lessons". In: (2021). DOI: 10.1145/3488932.3523256.

[5] Feng Hao and Peter Ryan. *J-PAKE: Authenticated Key Exchange Without PKI*. 2010. URL: moz-extension://4e2f15d4-655b-491c-9f2f-a029fbdc8002/enhanced-reader.html?openApp&pdf=https%3A%2F%2Feprint.iacr.org%2F2010%2F190.pdf.

[6] Feng Hao and Peter Y A Ryan. "Password Authenticated Key Exchange by Juggling". In: ().

[7] *J-PAKE protocol · J-PAKE over TLS*. URL: https://chunminchang.gitbooks.io/j-pake-over-tls/content/jpake/jpake.html.

[8] Imperial College London. *Discrete Logarithm Problem*. URL: https://www.doc.ic.ac.uk/~mrh/330tutor/ch06s02.html.

[9] Gregory Neven, Nigel P Smart, and Bogdan Warinschi. "Hash Function Requirements for Schnorr Signatures". In: ().

[10] Nick Sullivan. *A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography*. 2013. URL: https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/.

[11] *What Are Passive And Active Cyber Attacks? — RiskXchange*. URL: https://riskxchange.co/446/understanding-passive-vs-active-cyber-attacks-and-their-impact/.

[12] *What are Schnorr Signatures? - Bitstamp Learn Center*. URL: https://www.bitstamp.net/learn/blockchain/what-are-schnorr-signatures/.

[13] *What is a zero-knowledge proof and why is it useful?* URL: https://www.expressvpn.com/blog/zero-knowledge-proofs-explained/.

[14] *Zero-Knowledge Proofs Explained — ExpressVPN Blog*. URL: https://www.expressvpn.com/blog/zero-knowledge-proofs-explained-non-interactive-zero-knowledge-proofs/.