PEAR: A Hardware Based Protocol Authentication System

Sam Kerr, Michael S. Kirkpatrick, and Elisa Bertino
Department of Computer Science
Purdue University
West Lafayette, IN 47907, USA
stkerr@purdue.edu, {mkirkpat,bertino}@cs.purdue.edu

ABSTRACT

As users have to manage an increasing number of accounts, they have to balance password security and password usability. As such, many users use insecure passwords resulting in their accounts and data being vulnerable to unauthorized accesses. In this paper, we present Physically Enhanced Authentication Ring, or PEAR, a system that alleviates this problem. We leverage Physically Unclonable Functions (PUF) to create unclonable hardware devices, which users use to authenticate. Using a hardware device, our system uses zero-knowledge proofs, which provide better security than traditional passwords, yet users must only enter a simple PIN. As such, our system is very usable and imposes little to no burden on end users and service providers. We present transaction levels on top of PEAR of as an extension and then discuss some other work that could be done in the future.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*authentication*

General Terms

Security

Keywords

Authentication, Physically Unclonable Functions, Applied Cryptography, Hardware

1. INTRODUCTION

Users today have an increasing number of accounts and passwords that they must manage. Many service providers recommend creating a complex password containing numbers, punctuation, and no words in the dictionary. This would make it very difficult for an attacker to simply guess passwords. Ideally, every account a user has would have a different, secure password. In this way, if one of a user's accounts were compromised, no others would be in danger. In reality though, this does not happen.

Many users will not (or be able to) remember many different, secure passwords. If a user decides to create a secure password, he will probably use that password on many different accounts. Suddenly, if one account is compromised, many other accounts are in danger. Another problem is that since secure passwords are difficult to remember, many users will instead use very easy to guess passwords, such as a pet's name, a birthday, or even "password." Even worse, many users use these insecure passwords for all their accounts. In fact, a recent study [1] shows that when users use the same password in multiple locations, it is easier for attackers to recover a user's password to a higher security location, by attacking and stealing the user's password (which is the same for both sites) at a lower security site. This clearly shows a dangerous trend, which our system attempts to remedy.

Many users realize that they should use secure passwords and have a unique password for each service, but this is such a burden, that users fail to do so. What is needed is a system that does not burden the user with lots of complicated data to remember and is easy to use, at least comparable to typing a password into a keyboard, while at the same time providing a strong, different password for each user's account.

In this paper, we present Physically Enhance Authentication Ring, or PEAR, a system of protocols that allows users to use a simple PIN to securely manage the passwords to their various accounts securely. The user enters this PIN into a hardware device. The hardware device then automatically takes care of enrolling or authenticating a user securely using the protocols we have developed. Even though the user can use the same PIN for all accounts, the device creates a different a way to identify the user on every site, thus avoiding the problems discussed above. We leverage zero-knowledge proofs between service providers and the hardware device, so that even though the only input from the user is a simple PIN, the enrolment and authentication is more secure than a normal password.

A cornerstone of our protocols is the Physically Unclonable Functions (PUFs) technology. PUFs leverage randomness that was introduced through manufacturing or other ways to uniquely identify devices. For example, a PUF created using the same silicon wafer as another may have a length of wire that is one nanometer shorter than the other PUF due to manufacturing inconsistencies. This randomness is impossible to control and will slightly change the

SPRINGL '10, November 2, 2010, San Jose,CA,USA Copyright ©2010 ACM 978-1-4503-0435-1/10/11... $10.00.

behavior of the device. For every PUF, challenge-response pairings, denoted as (C_i, R_i), can be created. When executed with challenge C_i, a PUF can be uniquely identified by the R_i that it generates. Note that devices other than PUFs could be used, assuming that they provide this same sort of behavior. We refer to this component of the device as a Password Generator and Verifier (PGV).

The PEAR system will ensure that only the correct user with the PEAR device will be able to access the given services. In this way, a user can be confident that he will have privacy and that only he will be able to access his various PEAR enabled accounts.

We envision that our device and protocols will be used when interacting with websites that require stronger security than a typical website, such as a bank's website, health records system, or secure company database. For a user to sign up with one of these services, he will request to enroll and that service provider will send the user a hardware device for his use, as well as some data necessary for enrollment. However, the service provider will not choose the user's PIN, which ensures better security.

2. RELATED WORK

Secure authentication is a very common problem, and as such, has been explored extensively in the literature of computing [2, 3, 4, 5, 6, 7, 8]. While the security guarantees of these protocols may be strong under certain assumptions, there is a persistent underlying flaw that they fail to address. Successful authentication in these systems is based on the prover's ability to perform a computation based on a secret value. However, these protocols do not account for the fact that the prover consists of two distinct entities: a human user and his computer. As such, these protocols assume a trusted channel exists between the user and the machine that, in practice, does not exist. Furthermore, these solutions generally impose a large amount of overhead and make interactions unnecessarily complex for users.

Bumpy [9] is one approach to establishing this trusted channel. The system leverages a trusted hypervisor to detect a known key combination. When these keys are entered, the system switches to a mode in which keystrokes are encrypted. On the side of the computer, a TPM is used to ensure that only the current application can read the data, thus preventing malicious software from eavesdropping. However, as a special mouse and keyboard are required to perform the encryption, the user is tied to individual, Bumpy-enabled workstations.

Another approach to managing online credentials is that used in Microsoft CardSpace [10]. The intent of CardSpace is to allow users to disclose relevant credentials to website and service providers in a controlled manner. When using a CardSpace enabled service, the user selects which "Card" to provide, thus revealing the relevant information. While there are interesting aspects of this design, there are a number of problems. First, CardSpace is not universal. It has been installed starting with Windows Vista; earlier versions, such as XP, would require a manual download and install. The service is not available for other platforms. In addition, CardSpace assumes the presence of an uncorrupted version of the operating system. Our aim is to reduce the layer of trust to minimal components.

The main problem that exists with many current solutions is that they primarily rely on a software-based root-of-trust. Malicious software, such as a keylogger embedded in a rootkit, can either bypass the security mechanism or capture the user's password for future use. Although Bumpy raises the bar by including a hardware component, it relies on a trusted hypervisor to detect the special key sequence. Hypervisor rootkits have been demonstrated [11]; if such malware were installed below the trusted hypervisor, the security of Bumpy could be negated.

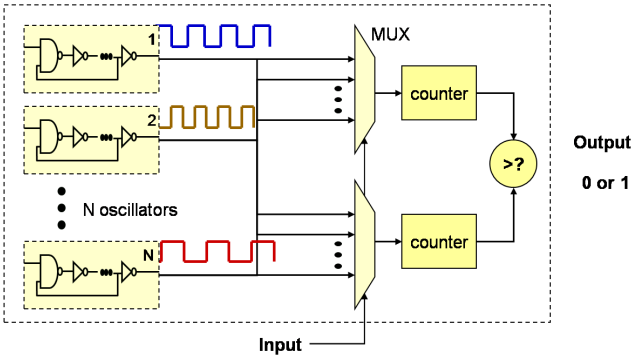


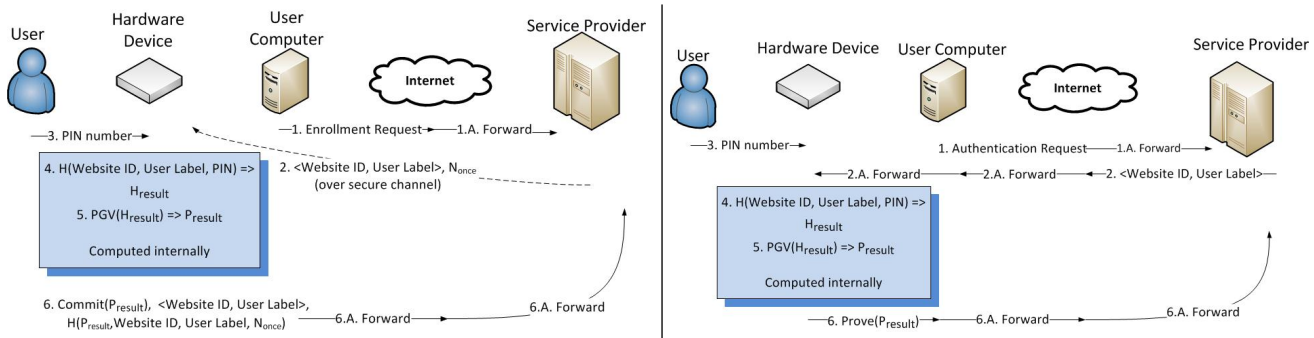
Figure 1: A simple 1-bit ring oscillator based PUF

We aim to counter these vulnerabilities by exploring the use of PUFs [12, 13, 14, 15, 16, 17, 18, 19]. The basic premise of a PUF is that the hardware creates a unique function that maps a challenge C_i to a response R_i. A commonly proposed application of PUFs is to provide secure cryptographic key storage. To protect a key K, the device can generate and store $K \oplus R_i = X$, which is a meaningless bit string that does not leak information about the key. The AEGIS secure co-processor [20, 21] has an integrated PUF for key storage and hardware-based randomness. A related body of work explores PUF-like behavior in RFID devices [22, 23, 24]. Our work contrasts with these previous approaches in that we are exploring new possible applications for PUFs that are not limited to key storage.

3. PHYSICALLY UNCLONABLE FUNCTIONS

An important part of the protocol is the use of Physically Unclonable Functions, or PUFs. A PUF is a device that takes an input and yields a response unique to both that challenge and to the individual PUF. That is, two PUFs given the same challenge will generate different responses. Because of this, PUFs are a good way to uniquely identify devices.

PUFs leverage randomness to generate these different responses. There are several different ways to create a PUF, such as shooting a laser through a film and examining the speckle pattern, which is what an optical PUF does, or running two signals through a circuit and seeing which signal reaches the end first, which is how an arbiter PUF functions. The PUF that we used is known as a ring oscillator PUF. This type of PUF leverages randomness introduced during the manufacturing process, due to manufacturing tolerances, to create unique responses. For instance, a wire may be specified as 100 nm long, but due to the manufacturing equipment, the wire may actually be 103 nm long or even 97 nm long. This variation is uncontrollable between individual products. Even though these devices have the exact same



(a) Enrollment Protocol

(b) Authentication Protocol

Figure 2: Protocols

design and were produced in the same batch, these small differences will change their execution. The PUF is designed to amplify these differences to create a new response to any given challenge.

Figure 1 shows an example of a 1-bit PUF based on ring oscillators. A ring oscillator is composed of an odd number of inverter gates, with the output of one being tied to the input of the next. This creates an oscillating signal. More inverter gates in the series will cause the oscillations to be slower, while fewer gates will increase the oscillations. Groupings of ring oscillators (as shown on the left of Figure 1) create a group of signals that oscillate at different rates. Using the input data, pairs of ring oscillators are selected and the number of oscillations is measured for a specific amount of time. The number of oscillations is stored in a counter, as shown in the middle of the figure. After a predetermined amount of time has passed, the two counters are compared. If the top counter is larger, then the output is a 1 and if the bottom counter is larger, then the output is a 0 . This forms the basis of a 1-bit PUF. Many 1-bit PUFs can be created and put on the same chip and combined to form PUFs that can produce longer bit length results.

A problem that arises with this type of PUF though is that there is a one-to-one correspondence between a changing input bit and a changing output bit. That is, on a 128-bit PUF, changing bit 54 of the input will change only bit 54 of the output. This could allow an attacker to model the behavior of the PUF, which would nullify any security benefit it had. A common technique to protect against this is to hash either the PUF's input or output. In this way, changing one bit of input will change multiple bits of output, making it nearly impossible for an attacker to model the PUF.

4. PROTOCOL DESCRIPTION AND ANALYSIS

In this section, we present the formal definition of our protocols. Initially, these are presented in a high level, implementation-independent way. In the following sections, we describe our implementation using concrete terms. For the protocols, we assume that authentication consists of a service provider, S, validating the user's claimed identity prior to providing the requested service. The client consists of the human user U, the trusted hardware device T, and the user's (untrusted) computer C.

4.1 Protocol Descriptions

Our protocols require the use of several cryptographic primitives. In the following sections, let $\mathsf{H}(\cdot)$ denote a collision-resistant hash function that is secure against probabilistic polynomial-time (PPT) adversaries. Let $\mathsf{PGV}(\cdot)$ denote a function that, given a challenge bit-string, will return a response bit-string unique to that challenge.

Initially, C will send a request to S initiating the protocol. S will respond to T with a tuple containing an ID to identify itself (such as its URL, if S is a website) and a unique ID for the specific user. During enrollment, S will send this tuple, along with a nonce N, via a secure channel, such as out-of-band communication or by encrypting the data with a public key for the device[1]. During authentication, the tuple, along with a one-time challenge, is forwarded through C. Once T has received the tuple, U physically interacts with T to enter a PIN. The physical interaction ensures that there is a trusted channel between the user and the device that does not depend on the computer.

After the previous steps have been completed, T now has the user's PIN, a means to identify the service, and the user's ID for that service. All subsequent client-side calculations are done directly on the device. T hashes the data, providing the result as input to the function $\mathsf{PGV}(\cdot)$. The output can now be used to uniquely identify the user's device, the service being requested, the user's ID on the service, as well as the user's PIN. The next part of the protocol is different depending on whether a user is enrolling or authenticating to S.

For enrollment, T computes a commitment to the output of $\mathsf{PGV}(\cdot)$. To bind this commitment with the tuple received from S, T hashes these pieces of data with N. As N is known only to S and T, the hash prevents a man-in-the-middle (MITM) attack. For authentication, T responds to the challenge from S by providing a zero-knowledge proof-of-knowledge of $\mathsf{PGV}(\cdot)$. S uses the commitment provided previously to evaluate the proof and validate the identity claimed.

Figure 3 formally defines our protocols. Let $\mathsf{Commit}(\cdot)$

[1]Note that, in the case of public key encryption, proper security requires that S have some means of confirming the validity of the key provided. For instance, T could be equipped with the ability to sign the key with a high level of assurance, such as using a TPM endorsement key. Addressing this problem is not trivial, but it is peripheral to our current focus. As such, we will not address it in detail.

Enroll(U) - Device T (using input data from user U) computes a commitment and enrolls the results with S.
- C requests enrollment from S - S sends the tuple <Label, ID> and nonce N to T over a secure channel - U sends PIN to T - T computes H(ID, Label, PIN) as H_{result} - T executes PGV(H_{result}) as P_{result} - T sends Commit(P_{result}), <Label, ID>, H(Commit(P_{result}),Label,ID,N) to S, via C
Authenticate(U) - Device T (using input data from user U) authenticates itself as a registered user of S.
- C initiates the authentication request from S - S sends the tuple <Label, ID> and Chal(P_{result}) to T - U sends PIN to T - T computes H(ID, Label, PIN) as H_{result} - T executes PGV(H_{result}) as P_{result} - T responds with Prove(P_{result}), which C forwards to S

Figure 3: Formal definition of protocols

denote a commitment scheme that ensure confidentiality against PPT adversaries. Chal$(\cdot)$ and Prove$(\cdot)$, indicate the challenge and response steps of a zero-knowledge proof for the value that was bound by the Commit$(\cdot)$ function. We assume that a PPT adversary, $\mathcal{A}$, has a negligible chance of guessing the results of Prove$(\cdot)$ without knowing the committed value.

In Enroll$(\cdot)$, the device is sent a tuple from the server S consisting of Label and ID. Label is used to identify the service itself. For a web service, this might be the server URL or some other unique piece of information about the server. The ID is used by the server to uniquely identify the user. Because of this, two users should not be able to have the same ID. However, due to the nature of the PGV, even if two users had identical IDs, they would commit different values. Note that this tuple must be sent over a secure channel, so as to prevent a MITM while enrollment is occurring. As we will show later, our protocol can defend against MITM during authentication, but during enrollment, the tuple must be sent securely.

The user also enters a PIN to the device, which ensures that a user is physically present and using the device. Requiring physical input from the user is important because even if an attacker was able to install a key logger or virus on the user's computer, he would not be able to recover the user's PIN. The user will interact with the device directly, without using the mouse or keyboard. As such, even if an attacker has compromised the computer and all input devices, we can still trust the channel between the user and device. Hence, an attacker would not be able to impersonate a user and execute the protocols. For added protection, the device could be equipped with a display reporting the Label and ID, thus providing the user a method to ensure that the enrollment is going to the intended recipient.

The tuple from S and the PIN from the user are then hashed together, directly on the device. The hash results are then used as input to PGV$(\cdot)$, which then creates the unique response. This response is the value that will be committed. It is important to note that since all of these calculations are done on the device itself, there is no way for an attacker to intercept any of the intermediate results, such as the hash results. Since an attacker would only be able to see the committed value coming out of the PUF, he would not be able to glean the necessary information to impersonate a user. The tuple the user was given is also sent to the service provider, which prevents an attacker from enrolling using a tuple he captured during another user's authentication session.

Next, depending on whether the user is enrolling or authenticating, using a zero-knowledge scheme, the PUF response is either committed to the server or a zero-knowledge proof is run, respectively. By using a zero-knowledge scheme, an attacker could intercept the traffic between the network and computer or between the computer and the device, but would not garner any useful information. Finally, by hashing Commit(P_result), the tuple, and the nonce, T is creating an integrity check for S. That is, the hash provides a means by which S can confirm that the tuple was received by T, and the commitment has been forwarded through C intact.

Authenticate$(\cdot)$ takes similar steps as enrollment. The server S sends a tuple containing S's label and the user's ID. The user then enters his or her PIN. These three pieces of data are then hashed on the device. The result of this hash is then used as input to the PGV device. Once the response is calculated, a zero-knowledge proof may be executed between S and T. After this proof is successfully completed, T has proven its identity to S and the user may gain access to S's services.

4.2 Security Analysis

We will now present a formal analysis of the security of our protocols. We begin by presenting several lemmas, which we then use as a basis to form a theorem that our protocols allow secure authentications. In our lemmas, we are considering the threat of a PPT adversary $\mathcal{A}$. If $\mathcal{A}$ is able to recover the PIN and construct a model of PGV$(\cdot)$ or capture the direct output of the PGV device (the "useful data" we mention in our lemmas), he would be able to impersonate the user with impunity. As such, our lemmas focus on proving the confidentiality of these pieces of data.

Lemma 1.
A man-in-the-middle attacker cannot recover any useful data communicated over the network between the service provider and the computer.
Proof: The only data that is transmitted between the computer and network is the tuple containing the service ID and the user's ID initially and then steps of the zero-knowledge proof (see Figures 2(a) and 2(b)). The tuple will only be sent during authentication, so we can assume that users are already enrolled. An attacker gains nothing by intercepting the tuple during authentication, since it still requires both

the user PIN number and the device itself to impersonate a user. Intercepting the steps of the zero-knowledge proof also gives him no information since these zero-knowledge protocols do not reveal any information about the committed value. □

Lemma 2.
A man-in-the-middle attacker cannot recover any useful data communicated between the user computer and the device.
Proof: As shown in Figures 2(a) and 2(b), the only data that is being transmitted over this channel is the tuple from the server and the zero-knowledge steps. As shown in Lemma 1, an attacker cannot gain any useful information from this. Also note that the PUF secret is never transferred outside of the device, but rather a commitment or proof is sent. As such, a MITM attack would not reveal the user's secret, but only the various steps of the zero-knowledge proofs, which are secure against MITM attacks. In addition, the service provider does not even ever know the user's PIN. □

Lemma 3.
An active man-in-the-middle attacker cannot recover any useful information by modifying data between the device and computer or computer and network during the authentication stage.
Proof: An attacker who modifies the tuple being sent to the device or computer from the network would cause the device to create an incorrect zero-knowledge proof. This would disrupt the user's ability to authenticate. However, the attacker would not be able to glean any information from the proof generated from this modified tuple, due to the use of the zero-knowledge proof. Note that an attacker would be able to recover useful information if it could modify the tuple during enrollment. It could substitute a malicious tuple for the valid tuple, which would cause users to be authenticating to the MITM, rather than the service provider. We avoid this problem by requiring that the tuple be sent securely during enrollment. □

Lemma 4.
A PPT adversary can impersonate a legitimate user to the server with only negligible probability.
Proof: As the final authentication step is to complete a zero-knowledge proof, an attacker would have to be able to defeat a zero-knowledge proof, which happens only with a negligible probability if an attacker does not know the user's secret. □

Lemma 5.
Given physical access to the device, an attacker could impersonate the legitimate user with only negligible probability.
Proof: If an attacker had access to the device, it would not be able to compute the proper hash value unless it supplied the correct PIN to the device. If the attacker attempted a brute force attack on the user's PIN, it would be trivial for a server to detect and disable the user's account temporarily. As long as the key space for the PIN is sufficient, this attack is not realistic. □

Lemma 6.
A legitimate user can authenticate to a legitimate S, except with negligible probability.
Proof: As a legitimate user would have access to the user PIN and a valid tuple from the server, he would be able to successfully complete the zero-knowledge proof, thus authenticating. □

Lemma 7.
An attacker cannot enroll using an existing or past user's credentials, except with negligible probability.
Proof: An attacker would be able to capture a user's tuple during authentication. It is plausible that he could attempt to enroll using this tuple. To prevent this, when the service provider issues the tuple initially, it also provides a nonce. During the enrollment protocol, the user submits the committed value, the tuple, and a hash of the tuple, nonce, and committed value. The service provider will verify that this tuple is valid. If the tuple is not valid or has already been enrolled, the service provider denies the enrollment request. □

Theorem 1.
The Enroll*(·) and* Authenticate*(·) portions of the protocol provide a secure means of hardware-based authentication.*
Proof: By Lemmas 1 and 2, we can be sure that any communication between the various parties is secure and will not leak information to an attacker. By Lemma 3, we can be sure that even if the device receives malicious data, it will not forfeit any sensitive data to an attacker. Using Lemma 4 and 5, we can also be confident that an attacker cannot impersonate the user, either by modifying network traffic or with access to the actual device. Lemma 6 shows that a legitimate user can successfully complete the zero-knowledge proofs. Lemma 7 shows that the system can protect against illegitimate users enrolling. Using these lemmas, we can conclude that a legitimate user can securely authenticate. □

4.3 Usability

One of the driving factors behind our work has been the observation that passwords are based on an out-dated attack model. Passwords were created when hardware costs were prohibitive and users had to share access to workstations. As the machine worked the same for all users, the burden of creating entropy was placed on the user in the form of complex password policies. In contrast, the current model is for each user to have a single-user workstation or laptop, accessing a remote shared server. As such, the burden of entropy creation should be passed from the user to the workstation.

Our design achieves this aim by reducing the amount of mental workload placed on the user. Instead of remembering a complex password, the user creates and remembers a short PIN. Furthermore, the user could apply the same PIN to all web sites, significantly reducing the mental workload. At this point, we can only make these claims intuitively. That is, proper evaluation of the usability requires a significant number of factors concerning the interface design. Our current aim is to evaluate the underlying protocols and design structure. We leave evaluating the usability claims for future work.

From the service provider's perspective, the main challenge is creating the secure channel during the enrollment phase. If out-of-band communication is used, this cost could be extensive. However, if the PGV device is equipped with a trusted public key infrastructure, this cost could be reduced to simply performing encryption. The storage requirements

on the server are essentially identical to current systems, as the service provider must maintain a list of user IDs and proofs of identity (a challenge-response system, rather than a password). In fact, our design greatly improves on the server's security, because the proofs in our protocol are significantly harder to guess than a user-created password.

5. IMPLEMENTATION

The implementation of the protocols above offered several different choices. There are several choices to make with regards to what algorithms should be used in the various steps, such as which hashing technique, commitment scheme, and zero-knowledge proof to use. After the algorithms were chosen, decisions needed to be made about the actual hardware that was to be used in the system and how the PUF would be implemented.

5.1 Algorithm Choices

The first choice to make was the hashing algorithm to use. We decided to use the SHA-256 hashing algorithm, as MD5 and SHA-1 have been cracked and are no longer valid options.

For the commitment and zero-knowledge proof algorithms, we used the Feige-Fiat-Shamir (FFS) scheme [25]. This scheme is relatively easy to implement and provides the necessary security for the protocols. The protocol involves a prover, P, and a verifier, V. FFS uses a Blum integer, n, as a modulus to perform its operations. Note that neither P nor V should know the factorization of n.

In order to use FFS for PUF-based authentication, P must first create a commitment to the PUF behavior. To do this, P computes PUF challenge-response pairs of the form (C_i, R_i) for $1 \leq i \leq k$, where k is a security parameter. The public commitment, then, consists of $R_1^2, \ldots, R_k^2 \pmod n$. Based on the intractability of computing modular square roots, PPT adversaries cannot recover the PUF responses from these public values. Based on these public values, the authencation process works as follows.

1. P chooses a random integer, r, a random sign $s \in \{-1, 1\}$, and computes $x \equiv s * r^2 \pmod n$. P sends x to V.

2. V chooses numbers $a_1, a_2, \ldots, a_k$ where a_i equals 0 or 1 and k is a parameter of the protocol. V then sends the bit string to P.

3. P computes $y \equiv r s_1^{a_1} s_2^{a_2} \ldots s_k^{a_k} \pmod n$, then sends this to V.

4. V verifies that $y^2 \equiv +/- x v_1^{a_1} v_2^{a_2} \ldots v_k^{a_k} \pmod n$.

If step 4 is true, V can be confident in P's identity. Note that this set of steps can be repeated as many times as necessary for V to be confident that P's identity is confirmed. Using PolarSSL [26], an embedded cryptography library, we were able to implement the FFS scheme on our devices. In our implementation, we repeated the protocol 4 times and set k = 5. This is a common choice of parameters for this protocol, as it ensures that an attacker can pose as P in less than 1 in 1 million attempts.

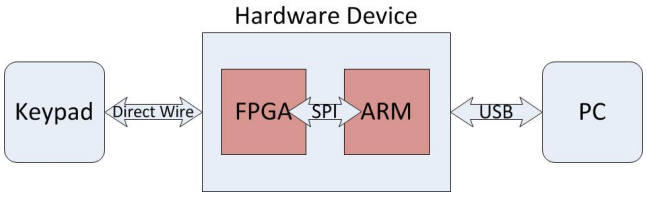


Figure 4: Macro view of implementation

5.2 Hardware Choices

Since we would be doing cryptography on our hardware device, this required not only that we found hardware capable of implementing a PUF, but also able to do the cryptographic heavy lifting. Because of this, we decided to use an FPGA for the PUF portion of the project and an ARM microcontroller for the communication and cryptographic steps. We used a Saxo-L board from KNJN, which features an Altera Cyclone FPGA and an NXP LPC2132 ARM chip. The two chips are connected directly to each other by a Serial Peripheral Interface, or SPI. There is also a USB port that was used to communicate with the computer. Figure 4 shows how the different parts of the protocol interact with the hardware device as well as how the chips in the hardware device interact internally. Note that even though we are using separate ARM and FPGA chips, they are both contained in the hardware device as a single piece.

In the past, PUFs have been implemented using FPGA chips. We decided to implement ours in the same way, since there are several PUF designs that are very easy to create on an FPGA, such as the one shown in Figure 1. We created independent pairs of ring oscillators to form a 1-bit PUF. Each oscillator consisted of nine inverting gates in series. Using fewer gates than this gave sporadic results, but we found nine to give consistent results.

We then fed the output from these ring oscillators to 32-bit counters. Using counters of this size ensures that we will avoid the problem of having the counters overflowing, since even with a 500 MHz oscillating signal from the ring oscillators, it would take over 8 seconds for the counters to overflow, but we are only running the oscillators for less than half a second. This counter size could probably be reduced to save space in the final chip.

Using this selection of parameters, we were able to implement a 32-bit PUF response on our Altera Cyclone. Ideally, this would be a larger value, such as 128-bit, so we ran the PUF several times, using the previous PUF response as the next PUF challenge until we had generated a 128-bit response. In a production environment, a larger FPGA would be chosen so that a 128-bit PUF could be created directly.

The ARM chip allowed us to use libraries written in C, so we used the PolarSSL library to implement the cryptographic steps. The ARM also handled all of the I/O with the USB controller and with the user. Some ARM cores actually can be embedded inside of FPGAs directly, which would have simplified our design greatly. This would be something to investigate when creating a production quality device, so that only 1 chip was required.

For I/O with the user, we connected the Saxo-L board to a breadboard, which had several DIP switches that a user used to input his PIN. In a production setup, the I/O could

be a keypad that is soldered to the device, providing a more resilient, permanent setup than a breadboard.

The board then communicated with the PC and sent the necessary data to a custom Windows application. Users would enter the PIN and receive their data for steps of the protocols through this application. In the future, it might be interesting to create a Firefox plug-in so that a user could use the system directly in the browser. For applications that are not browser based, we could provide some sort of API that would allow programs to handle all the interaction programmatically.

Another step that could be taken for a production quality device is to combine the FPGA and ARM as a single chip. Taking this step would simplify the device, since it would only require one chip rather than two and the circuitry to interconnect them. To do this, we would select a larger FPGA and program an ARM core onto it. Then we could implement the SPI connection internally. This is fairly common and has been done before. Another alternative is to use a system such as AEGIS, which is an ARM processor with a PUF device embedded into it, allowing much more efficient PUF to ARM communications.

Due to the size of the equipment involved, a PEAR device could be made very compact and portable. This would allow users greater mobility, since they could carry a PEAR device on their person easily and use it at multiple PEAR enabled workstations. This is much better than a user being tied to a single workstation, especially since the PEAR system would be used for various web applications, which a user may need to access from multiple locations. Creating the physical PEAR device for production usage will certainly require that it is made in a way to ensure that users' mobility is not obstructed by the system.

6. EXTENSIONS

A common source of security vulnerabilities in web-based environments is that all session actions are treated equally. For instance, a typical web-based application will require authentication only at the beginning of the session. All subsequent requests, whether they involve reading data or modifying data, are granted, as the user is logged in. A more sophisticated design would acknowledge differences in "transaction levels." Intuitively, a transaction level is a classification describing the sensitivity of the request.

In the context of a banking application, one transaction level could consist of non-desctructive actions, such as checking one's account balance. Another level could consist of recoverable, audited transactions, such as transferring money between accounts owned by the same user. One more could consist of transfers between accounts within the same bank. While a final could be for transfers to external accounts.

In our design, we see a number of extensions to our protocols to support transaction levels. A preliminary approach could be for the service provider to specify re-authentication policies. That is, when the user attempts to perform a sensitive actions that change settings, the server could require the user to re-authenticate. As the authentication protocol requires physical interaction with the user, malware installed on the user's computer would not be able to initiate the request illicitly.

Alternatively, the service provider could require different challenge-response pairs for different transaction levels. One way to do this would be to require the user maintain different PINs for different levels. As a result, each level would have a different H_{result}, which would change the input to the $\mathsf{PGV}(\cdot)$ function. Another approach, which would be transparent to the user, would be for the service provider to use a different Label or ID for different levels for the same user. Just as above, this would change H_{result}, which would affect the challenge-response behavior.

Properly implementing transaction levels would require addressing a number of technical challenges. First, we would need an appropriate policy language to capture the behavior. Next, we would need to explore how to present the user with a usable interface that abstracts the behavior adequately. Finally, we would need to consider the trade-offs in usability and security that would result from applying the approaches described above. We leave such details for further investigation.

7. CONCLUSIONS

We have presented PEAR, a system that will allow users to authenticate in a way that is secure, yet avoids the common problems that affect many other systems. Our system's biggest strength lies in the fact that it uses an external hardware device, which receives both data from a service provider and a PIN entered by the user. Because the PIN is entered only into the device and not into the user's computer, an attacker has no way to recover it.

Additionally, our protocols use zero-knowledge proofs to generate commitments and proofs for communication between the service provider and the hardware device, using the computer as an intermediary channel. Because of the zero-knowledge proofs though, a MITM attack is not possible, either from the user's computer or over the network between the computer and the service provider. The device also executes all cryptographic functions and calculations internally, which further prevents an attacker from recovering any useful data.

Besides just providing a very secure system for users to authenticate themselves, our system also provides a great amount of usability in every day scenarios. In a normal authentication scheme, users would type a password on a keyboard. In our system however, they simply plug the device into the computer and type a PIN. This places very little additional burden on users and is very usable. Little burden is placed on to service providers as well.

Currently, our system ensures a large amount of security between a user and a single service provider. However, users may need to enroll in more than one service. It would be convenient for them to use the same device and PIN for each different service provider. This is currently possible, since each service provider will give a different tuple to the user. However, this approach still requires that users execute the enrollment procedure at least once with each different service provider. In the future, it would be interesting to investigate a way for users to use their existing accounts and enrollments to log in to other services, ones which they have not enrolled with. This system would lessen the burden on users, since they would not have to worry about enrolling multiple times at different sites. This would pose various difficulties though. An example is if the one service utilizes transaction levels while the other does not and how to resolve this. Also, it is important not to leak any information about a user during the communication. In addition, a formal protocol language would need to be defined. We leave the solution to these

problems as future work.

8. ACKNOWLEDGEMENT

The work reported in this paper has been partially funded by Sypris Electronics.

9. REFERENCES

[1] J. Bonneau and S. Preibusch, "The password thicket: technical and market failures in human authentication on the web," in *Ninth Workshop on the Economics of Information Security (WEIS)*, 2010.

[2] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, December 1978.

[3] D. J. Otway and O. Rees, "Efficient and timely mutual authentication," *ACM SIGOPS Operating Systems Review*, vol. 21, pp. 8–10, January 1987.

[4] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, "Kerberos authentication and authorization system," in *Project Athena Technical Plan*, 1987.

[5] T. Y. Woo and S. S. Lam, "Authentication for distributed systems," vol. 25, no. 1, pp. 39–52, January 1992.

[6] J. Clark, "Attacking authentication protocols," *High Integrity Systems*, vol. 1, 1996.

[7] J. Heather, G. Lowe, and S. Schneider, "How to prevent type flaw attacks on security protocols," in *Proceedings of the 13th IEEE Workshop on Computer Security Foundations (CSFW '00)*, 2000.

[8] S. Malladi, J. Alves-foss, and R. B. Heckendorn, "On preventing replay attacks on security protocols," in *Proceedings of the International Conference on Security and Management*. CSREA Press, 2002, pp. 77–83.

[9] J. M. McCune, A. Perrig, and M. K. Reiter, "Safe passage for passwords and other sensitive data," in *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, Feb. 2009.

[10] D. Chappell, "Introducing windows cardspace," http://msdn.microsoft.com/en-us/library/aa480189.aspx, 2006.

[11] S. T. King, P. M. Chen, Y. min Wang, C. Verbowski, H. J. Wang, and J. R. Lorch, "Subvirt: Implementing malware with virtual machines," in *In IEEE Symposium on Security and Privacy*, 2006, pp. 314–327.

[12] G. E. Suh and S. Devadas, "Physcal unclonable functions for device authentication and secret key generation," in *Proceedings of the 44th IEEE Design Automation Conference (DAC)*. IEEE Press, 2007, pp. 9–14.

[13] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled physical random functions," in *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*, 2002.

[14] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "Physical unclonable functions and public-key crypto for FPGA IP protection," in *International Conference on Field Programmable Logic and Applications*, 2007, pp. 189–195.

[15] K. B. Frikken, M. Blanton, and M. J. Atallah, "Robust authentication using physically unclonable functions," in *Information Security Conference (ISC)*, September 2009.

[16] K. Lofstrom, W. Daasch, and D. Taylor, "IC identification circuit using device mismatch," in *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*, 2000, pp. 372–373.

[17] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS '02)*, 2002.

[18] S. Stanzione and G. Iannaccone, "Silicon physical unclonable function resistant to a 10^{25}-trial brute force attack in 90 nm cmos," in *Symposium on VLSI Circuits*, 2009, pp. 116–117.

[19] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, pp. 2026–2030, September 2002.

[20] G. E. Suh, C. W. O'Donnell, and S. Devadas, "Aegis: A single-chip secure processor," *IEEE Design and Test of Computers*, vol. 24, no. 6, pp. 570–580, 2007.

[21] ——, "AEGIS: A single-chip secure processor," in *Elsevier Information Security Technical Report*, vol. 10, 2005, pp. 63–73.

[22] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola, and V. Khandelwal, "Design and implementation of PUF-based "unclonable" RFID ICs for anti-counterfeiting and security applications," in *2008 IEEE International Conference on RFID*, 2008, pp. 58–64.

[23] B. Danev, T. S. Heydt-Benjamin, and S. Čapkun, "Physical-layer identification of RFID devices," in *Proceedings of the USENIX Security Symposium*, 2009.

[24] N. Saparkhojayev and D. R. Thompson, "Matching electronic fingerprints of RFID tags using the hotelling's algorithm," in *IEEE Sensors Applications Symposium (SAS)*, February 2009.

[25] U. Feige, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, 1987, pp. 210–217.

[26] "Polarssl: Small cryptographic library," http://www.polarssl.org/, 2008.