SCIENTIA V.12Nº1 (2001) 11-34

KHAP: Using Keyed Hard AI Problems to Secure Human Interfaces

Jeff King[1], Andre dos Santos[1], Chaoting Xuan[1]

[1]College of Computing – Georgia Institute of Technology*
801 Atlantic Dr – Atlanta, GA, 30332-0280, USA
{peff,andre,cxuan}@cc.gatech.edu

ABSTRACT

There is often a need for users to securely interact with a remote computing system, ensuring the integrity and authenticity of transmitted messages. Typical solutions assume that a local trusted computing platform is available to perform cryptographic operations, but this is often not the case. We introduce KHAP, a protocol for using hard artificial intelligence problems to provide message authentication checks centered around a human verifier. We also formally introduce the notion of a keyed hard AI problem, which is one that uses an authentication key to prove the source and integrity of a message. We give examples of some keyed hard AI problems, as well as examples of KHAP's applicability to the specific problem domains of Internet voting and the use of smartcards for digital signatures.

KEYWORDS Authentication, Human Verification

*This work was supported by a grant from Microsoft Corporation.

ISSN 0104 – 1770

1 Introduction

Suppose Alice is on a trip to a computer security conference. Her coworker, Bob, stays behind to watch over their cubicles. Suddenly, Alice remembers that she forgot to send an important memo to Bob. No problem, she thinks – there are computers available at the conference, and she can just fire off an email to Bob. Because it's so important, Alice is going to sign the memo. Because Alice always carries a smartcard that has her private key, she can sign the memo. She sits down at a conference computer, plugs her smartcard into the computer's reader, and types out the memo. She presses the send button, and the memo is sent to her card for signing; the signature comes back to the computer and is emailed with the memo to Bob. Or is it? What did the smartcard really sign? How does Alice know it's the same thing that was on her screen? How does Alice even know if anything was sent to her card?

When humans interact with computing systems, the human often wants to know that data has reached the intended system, and that it has not been modified in transit. Traditionally, this issue has been dealt with by making one of two assumptions:

1. there is a direct channel between the human and the system; the human identifies the system by its ability to transmit on the channel, and it is assumed that an attacker is not able to tamper with data on the channel. Examples include Automated Teller Machines (ATM) and programs running locally on Personal Computers (PCs).

2. the human has a direct channel to a computing platform that he trusts at least as much as the destination system. This trusted platform communicates with the destination system over an untrusted channel; it uses cryptography to verify the system's identity and the integrity of the data. An example is using a PC to browse a web page through an encrypted

tunnel.

In our example, Alice assumed that she had a direct, secure link to her smartcard. In reality, she was trusting the computer she was sitting at to accurately relay the information. If the computer was acting maliciously, it could easily have displayed one memo on screen and sent another to the smartcard to be signed; Alice has no way of directly communicating with the smartcard.

There are many cases where a computing system is not accessible by a direct, trusted link. The system may be physically located far away. It may be incapable of direct interaction, as with a typical smartcard. Likewise, the assumption that an intermediate platform can be trusted is often false. The platform may be a public system, owned and controlled by an untrusted third party. Even if the owner is trusted, the platform may be infected by a virus or trojan horse program. It may be possible to insert a "false front" between the user and the system (e.g., a physical device between a human and ATM that records account and PIN numbers).

There is therefore a clear need for a system to allow secure interactions with a remote trusted system. In this paper, we introduce KHAP, a protocol model that allows a human to securely interact with a remote computing system without making either of the two assumptions listed above. The model is based on using hard artificial intelligence (AI) problems to transform data into a form that humans can easily decipher, but which computers have difficulty interpreting.

Section 2 discusses existing work in this area and defines the building blocks of the protocol. Section 3 describes the protocol itself, while section 4 provides more concrete examples of KHAP parameters. Security and human usability properties are discussed in sections 5 and 6. Some more complete examples are given in 7. Finally, we indicate future directions and conclusions in sections 8 and 9.

2 Background

This section describes existing work and defines the building blocks of the KHAP protocol.

2.1 Related Work

The problem of malicious terminals has been examined many times. However, the focus is often on the identification of humans to computing systems (Matsumoto and Imai 1991; Hopper and Blum 2001). Recently there has been more development of systems focusing on humans verifying computers without the aid of a trusted computing system.

Naor and Pinkas propose the use of visual cryptography for message authentication (Naor and Pinkas 1997). The user carries a printed transparency as a key; the cipher text is an image of the same size. When a user places the transparency over the image, the plaintext is revealed (with some amount of random noise). Some of the problems with this system include the necessity of carrying the transparency, and the fact that the transparency can only be used securely a very small number of times.

Berta and Vajda propose a system of "biometric signing" (Berta and Vajda 2003); this system is focused on sending signed messages from an untrusted terminal. Their model focuses on a remote sender verifying a signature; however, it is often useful for the sender to verify that his message reached the destination intact.

Gobioff et al describe a protocol for using a low-bandwidth (e.g., single-bit) secure channel to achieve more complex operations (Gobioff, Smith, Tygar, and Yee 1996). They suggest changing the traditional smart card design to accommodate this secure channel. However, this suggestion has not been adopted in practice, and is not feasible for problem domains other than smartcards.

Recent research has attempted to exploit the intelligence gap between humans and machines. A Human Interactive Proof (HIP) is a method by which a computer can tell a human and a machine apart. HIP research is mainly motivated by efforts to defend Web services from abuse by programs. A formalization of AI problems as security primitives can be found in (von Ahn, Blum, Hopper, and Langford 2003).

2.2 Encryption

Our protocol model utilizes a pair of encryption/decryption functions that can be performed by a human. These functions are denoted E and D respectively. Let P be the set of all plaintext messages, C be the set of all encrypted messages, and K be the set of all keys. E is defined as a function mapping keys and plaintext to ciphertext: $E : P \times K \rightarrow C$. The function D performs the inverse: $D : C \times K \rightarrow P$, with $D(E(p,k),k) = p$ for all $k \in K$ and $p \in P$. Furthermore, we require that it be impossible to accurately guess the ciphertext that a given plaintext will generate if the key is not known. That is, for every $c \in C$ and every $p \in P$ and every $k \in K$, the probability $Pr[E(p,k) = c]$ is $\frac{1}{|C|}$.

We will also consider the concept of the validity of an encrypted message. A given message c is valid for a key k if and only if $c \in C$ and there exists a plaintext p such that $E(p,k) = c$. In other words, if a given message is supposedly the result of an encryption with a given key, then there must be a possible plaintext. We use the function $V(m,k) = \{true, false\}$ to denote whether a message m is valid for a given key k.

2.3 Hard AI Problems

The use of hard AI problems as security primitives was introduced in (von Ahn, Blum, Hopper, and Langford 2003). We will reiterate their definitions of AI problems and hard AI problems, and

we will provide a definition for a specific subset of AI problems, which we call Transformable AI Problems (TAP).

Definition 1 An AI problem is a triple $P = (I, R, f)$ where I is a set of problem instances, R is a probability distribution over the problem set I, and $f : I \to \{0,1\}^*$ answers the instances. Let $\delta \in (0,1]$. For an $\alpha > 0$ fraction of humans H, $Pr_{x \leftarrow R}[H_r(x) = f(x)] \geq \delta$, where $H_r(x)$ represents a human attempt at answering the instance, and $x \leftarrow R$ indicates the selection of x from the distribution R.

Definition 2 An AI problem P is said to be (δ, τ)-hard if there does not exist a program A running in time at most τ on any input from I, such that

$$Pr_{x \leftarrow D}[A(x) = f(x)] \geq \delta$$

The program A is considered to be a program running on modern computer hardware using state-of-the-art algorithms.

Definition 3 A Transformable AI Problem (TAP) is an AI problem with the additional constraint that the hardness must come from deciphering a data transformation. A given TAP problem has a set of transformations T, and a set of input messages M. Recall that an AI problem is given by the triple (I, R, f). The set of problem instances I is defined as the set of all possible transformations: $\{t(m) : t \in T \text{ and } m \in M\}$. The instance distribution R is again a distribution over I. The answer function, f, converts a problem instance back to the original input: $f(t(m)) = m$ for all $t \in T$, $m \in M$.

If a TAP problem is (δ, τ)-hard, then the transformation itself is (δ, τ)-hard:

$$Pr_{(m,t) \leftarrow R}[H_r(t(m)) = m] \geq \delta$$

$$Pr_{(m,t)\leftarrow R}[A(t(m)) = m] < \delta$$

That is, recovery of the original message from the transformed message has at least δ probability for humans, but less than δ for current programs. Note that for some problems, computers may perform similarly to humans (i.e., the probabilities for success both approach δ). In this case, the gap between human computer success can be amplified; techniques are given in (von Ahn, Blum, Hopper, and Langford 2003).

2.4 Keyed Hard AI Problems

A keyed transformation is a data transformation that takes an additional key parameter, k. A human with knowledge of k should be able to recognize the presence or absence of the key in the transformed value. We denote the human process of distinguishing two transformed values as a function H_d. The value $H_d(t_1,t_2)$ is true if and only if a human can tell that t_1 and t_2 are different values.

Definition 4 A keyed transformation T is said to be $(\alpha,\varepsilon,\tau)$-distinguishable for a human H if, for any two keys k and k' whose absolute difference $|k-k'| > \varepsilon$ and for any $m \in M$

$$Pr[H_d(T(m,k),T(m,k'))] \geq \alpha$$

and if there does not exist a program A that runs in time τ such that, given $m \in M$ and $t = T(m,k)$ and $m' \neq m$

$$Pr[H_d(t,A(m',t))] < \alpha$$

That is, a human can tell with probability α that two messages were transformed with sufficiently different keys. Furthermore, there is no current program to convert a transformed message into a different message without creating two distinguishable values.

Note that we use the notation $|k-k'|$ to refer to the absolute difference between two keys; however, the "value" of the keys will often not be easily quantifiable. In such a case, the ability to distinguish between two keys can be determined experimentally (i.e., we care only that the keys are "sufficiently different" for our purposes, not about their actual values).

Definition 5 A TAP problem is $(\alpha,\varepsilon,\tau)$-keyed if every transformation $t \in T$ is $(\alpha,\varepsilon,\tau)$-distinguishable.

3 KHAP Protocol Model

The participants in this model are the human, H, the remote computing system S, and a man-in-the-middle attacker, MM. It is assumed that H and S are able to communicate with each other, but that MM may block or modify messages without the knowledge of H and S. It is also assumed that H and S may agree on a secret beforehand.

Furthermore, H and S have agreed on two functions E and D, which match the encryption and decryption functions described in section 2.2. They have also agreed upon a TAP problem that is (δ,τ)-hard and $(\alpha,\varepsilon,\tau)$-keyed. We use $T(m,k)$ to denote the computation of a problem instance wherein a transformation is chosen randomly from the set of transformations and applied to m using key k. The functions H_r (human extraction of a message from its transformed state), H_d (human distinguishing of two transformed messages), and V (verification of an encrypted message) all have the same meanings as given in the previous sections.

The security goals are to allow H to verify with high probability that a given message originated with S, that messages from S have arrived unmodified, and that messages sent to S have arrived unmodified. The notion of "high probability" in the goals

is dependent on the hardness and keying parameters given above. When choosing a TAP problem, participants should consider the parameters of the problem in relation to the desired level of assurance. Note that confidentiality of the transmitted information is not a security goal.

The protocol behavior is described in the following sections. Each section details the behavior of the two participants, H and S, in a given situation.

3.1 S transmits to H

Assume that S has a message m_0 that it wants to transmit to H. S and H share two secrets, k_1 and k_2. H is expecting the message. The behavior of S is as follows:

1. S computes $m_1 = E(m_0, k_1)$
2. S computes $m_2 = T(m_1, k_2)$
3. S transmits m_2 to H

The behavior of H is as follows:

1. H receives m'_2; if message does not arrive within a timeout period, H assumes that message has been lost
2. H computes $m'_1 = H_r(m'_2)$
3. H computes $v_T =$ logical negation of $H_d(m'_2, T(m'_1, k_2)))$
4. H computes $v_E = V(m'_1, k_1)$
5. H computes $m'_0 = D(m'_1, k_1)$
6. H believes the message to have originated with S and have arrived unmodified if and only if both v_T and v_E are true.

3.2 H transmits to S

Assume that H has a message m that he wants to send to S. S and H share two secrets, k_1 and k_2.
The behavior of H is as follows:

1. H transmits m to S

2. H waits for a response r from S; if response does not occur within a timeout period, H assumes that message did not reach S

3. H verifies r as m_2 in the previous protocol; if verification fails, H assumes that message did not reach S intact

4. H compares $m' = D(H_r(r), k_1)$ to m; if not equal, H assumes that message did not reach S intact

The behavior of S is as follows:

1. S receives m'

2. S computes $r = T(E(m', k_1), k_2)$

3. S transmits r to H

4 Examples of Functions

In this section, more concrete examples for E and T are given. These lists are not meant to be exhaustive, but to provide examples that we believe meet the definitions listed in section 2.

4.1 Encryption Functions

The exact encryption function used will depend largely on the type of messages to be encrypted. In all cases, the size of the keyspace will be very short compared to traditional cryptography; this is a requirement since a human must be able to remember the key and perform the decryption manually.

Substitution One simple encryption function is message substitution. That is, one message is substituted for another according to a key. In order for this to be feasible for humans, the number of messages must be very small; furthermore, the mapping of messages must be simple to remember. The former requirement is subject to the constraints of the problem domain. The latter requirement can be helped with the use of semantic mappings. Consider an example where $M = \{yes, no\}$ and K consists of pairs of sets of words with semantic similarity. So if $k =$ (fruits, cities), then $E(yes, k)$ might be "apples" and $E(no, k)$ might be "Atlanta."

Null Encryption As a special case, we will examine the security impacts of using the identity function ($E(m,k) = m$ for all m and k). This function has the advantage of requiring no effort on the part of the user. It is also more flexible, in that the set of messages does not need to be known beforehand.

4.2 TAP and Keyed TAP Problems

This section contains problems we believe to meet the requirements for TAP and keyed TAP problems. Because TAP problems derive their hardness from a set of transformation functions, the problems are described in terms of their transformations.

Speech Synthesis One possible scheme is to produce audible human speech; that is, if m is text, then $T(m,k)$ is a digital sound

file containing the spoken text of m. Furthermore, k is a set of values modeling the vocal properties of a speaker such that for some ε, T is $(\alpha, \varepsilon, \tau)$-distinguishable.

The key is a representation of a human vocal tract. This is typically represented as a plot of frequency against time called a cepstrum. There is significant research in the area of modeling vocal parameters (Monrose, Reiter, Li, and Wetzel 2001).

To generate a key, S chooses random values within the vocal model (within some parameters that still represent human voice). H is "trained" on the voice by listening to several samples produced by it. Note that H does not have specific knowledge of the key, since it would be difficult to remember and useless for manual computation. However, the memory of the voice allows H to perform the distinguishability test.

When S wishes to compute $T(m,k)$, it first synthesizes the voice based on the model given by k (H must have been previously trained to recognize k). Then it may apply a filtering transformation that adds noise to the resulting audio file. Examples of such transformations are described in (Kochanski, Lopresti, and Shih 2002). The purpose of the noise is to mask the speech in such a way that humans can still understand and recognize it, but computer speech recognition systems are thwarted.

A human may compute $H_r(m)$ merely by listening to the audio file and understanding the spoken words. H_d consists of correctly identifying the vocal properties of the speaker.

3D Rendering A three dimensional (3D) rendering problem is a TAP problem defined by the tuple (L,S,N,F) where L is a set of scene names, S is a set of 3D renderable scenes, N is a function mapping each scene name $l \in L$ to a set of scenes $s \subset S$, and F is a function mapping text $m \in M$ to a 3D renderable version of the text.

The key k is an ordered list of scene names. To compute $T(m,k)$, the remote computing system S performs the following

algorithm:

```
foreach l in k do
  randomly choose a scene, s, from a uniform distribution
    over the set N(l)
  insert F(m) into the scene
  randomly choose a camera location within the scene
  render the scene
done
```

$T(m,k)$ returns the ordered list of rendered scenes. H must be able to recognize the scene name based on a rendering of the scene; H can distinguish messages produced by different keys by the presence and ordering of specific scenes (he must therefore remember the ordering of the scenes in his key). Computing $H_r(m)$ consists of reading the text found within the scene.

For example, consider a key that consists of one label, "house." The remote computing system S randomly selects a scene it knows about that matches "house." It inserts a 3-D version of the text into the scene and raytraces the result to a 2-D image. When H sees the image, he extracts the text by reading it. He verifies the key by confirming that the scene is one of a house.

In order for an attacker to compute $A(T(m))$, he must be able to identify the text within the scene, convert it to its original model, and perform the inverse of F.

Keyless Transformation Some problems believed to be hard TAP problems are already known; these are described in (Kochanski, Lopresti, and Shih 2002; von Ahn, Blum, Hopper, and Langford 2003). We will examine the security implications of using keyless TAP problems with KHAP. These functions have the advantage that they are already being analyzed and used in production systems. They also may have a stronger hardness guarantee and may put less burden on the user.

5 Security Analysis

This section considers some possible attacks by *MM* and describes the role of the E and T functions.

5.1 *MM* discards messages

It is always possible for *MM* to discard messages and perform a Denial of Service attack. However, H assumes the worst in the absence of an expected message. It is therefore not possible by discarding a message for *MM* to trick H into thinking that a message was successfully delivered. In the case of unexpected messages (from S to H), it is possible to silently discard the message without H realizing. It is assumed that the underlying message protocol will perform in such a way that S does not send unexpected messages, or that missing such messages will not be a serious security breach.

5.2 *MM* modifies message in transit

Assume that H sends a message to S, and *MM* changes the message m in transit to m'. S returns the transformed message r' such that $r' = T(E(m', k_1), k_2)$. *MM* intercepts r' and must send r to H such that $H_d(r, T(E(m, k_1), k_2))$ is false and $D(H_r(r'), k_1) = m$. That is, *MM* must compute a value for r such that H is fooled into thinking it was computed by S. Furthermore, the computed r must correctly decrypt to the original message sent by H. *MM* knows r' and m because it intercepted them in transit. It also knows m', since that is the forged message it has created.

We will examine the implications when using keyed and regular TAP problems in conjunction with null and non-null encryption functions.

Keyed Transformation and Non-Null Encryption Assume that the participants are using a keyed hard TAP problem and non-identity encryption. *MM*'s attack consists of the following steps:

1. *MM* attempts to break E through cryptanalysis over multiple runs. Because the encryption key is short enough for a human to use, it is likely that cryptanalysis will be fruitful if enough plaintext-ciphertext pairs are captured and recognized by *MM*. For some problem domains, *MM* can potentially stimulate multiple runs by repeatedly sending fake requests to S.

   The plaintext m is known to *MM*. However, the ciphertext $c = E(m,k_1)$ may be difficult to acquire, since it requires $c = A(T(c))$ (that is, reversing the AI transformation), which is believed to be hard. Some instances where it may be possible to recognize the ciphertext include:

   - A malicious human may compute $H_r(T(c))$ and feed the result to a computer for cryptanalysis.
   - *MM* may have a low but statistically significant probability of computing $A(T(c))$. Over a sufficient number of attempts, enough data may be collected for successful cryptanalysis. The exact details depend on the (δ,τ)-hard parameter of the specific transformation used.

2. If E is broken, then *MM* may easily compute $c = E(m,k_1)$ without knowing k_1. However, he still must compute r such that it is indistinguishable from $T(c,k_2)$. But following the definition of a keyed transformation (section 2.4), no program exists to compute r with probability greater than α.

In order to succeed in fooling H, *MM* must successfully complete both steps.

Null Encryption If the identity function is used for encryption, then clearly the first step becomes trivial. *MM* needs only to complete the second step, producing a keyed transformation that fools H.

Non-Keyed Transformation If a regular TAP problem is used to provide the transformation step, then the second step becomes trivial. The integrity of the system relies solely on the inability of *MM* to break the encryption function, as shown in the first step.

Non-Keyed and Null If a keyless transformation is used with the identity function, then both steps become trivial. In this case, the protocol is providing no assurance whatsoever of the integrity of messages.

S **sends to** H If we assume that *MM* is able to guess the message, then *MM*'s task is identical to the one described above, with one caveat: H does not already know the contents of the message. Therefore *MM* does not have to construct r such that $D(H_r(r),k_1) = m$, but rather such that $V(H_r(r),k_1)$ is true (that is, such that the message is a possible, valid one). This is still not trivial, but allows the possibility that rather than breaking E, *MM* sends random (possibly valid) messages. This is not likely to be a problem, as a human receiving a string of unintelligible messages will presumably become suspicious.

5.3 Replay Attacks

MM may mount a replay attack by saving valid messages sent between H and S and resending them later. The duplicate messages may be used to repeat operations or may have an unexpected meaning in a different context.

KHAP doesn't directly deal with replay attacks; however, it is simple to incorporate prevention measures into a protocol that

uses KHAP. For messages from *S* to *H*, *S* can insert a timestamp into outgoing messages that is authenticated along with the message contents. The timestamp may be based on either wall clock time or a monotonically increasing arbitrary value.

In the case of an arbitrary value, *H* must keep track of the value and verify that it increases in each message. In order to prevent replay attacks across sessions, *H* must remember the current value until the next session. This may be prohibitively difficult for most humans.

Using wall clock time provides a convenient global timer. The clock must not reset, so it must contain the full date and time to a resolution of *t* time units. *H* confirms that the timestamp on each message is increasing; therefore *S* cannot send more than one message in each *t* interval. *H* must also confirm that the timestamp is within *s* units of his current idea of the time (based on an independent clock). While some clock skew is inevitable between *S* and *H*, the skew can be overlooked by *H* if it is smaller than *s*.

In the case that an attacker replays a message from *H* to *S*, the situation is somewhat more complex. *H* will not be expecting any messages from *S*, and so cannot participate in the verification. In fact, KHAP takes no precautions to prevent arbitrary forged messages being sent to *S*. It is up to *S* to use a different mechanism to authenticate messages from *H*, such as a traditional password; the use of such methods is outside the scope of this paper. On the other hand, if *S* is a device that can be turned off, like a smartcard, these attacks cannot happen arbitrarily. They can occur only when the card is on, such as when the smartcard is plugged in to a malicious terminal.

6 Human Requirements

One of the most important aspects of this system is usability by humans. The behavioral description of the protocol spells out several steps for the user to perform; however, most of these steps should occur naturally and easily (without the need to follow a memorized script).

The human must be able to "time out" when messages fail to arrive; it is natural for users to expect computers to react reasonably quickly. If a response does not arrive within what a human considers a reasonable time, it can be considered to have timed out.

The human must also be able to compute $H_r(m)$ for a given message m. The definition of an AI problem gives at least δ probability of success for a human. In practice, this step may consist of operations as simple as reading or listening to the message m.

The human must be able to distinguish between messages transformed with the correct key, and those that have been forged. The definition of a distinguishable transformation gives at least α probability of success in distinguishing correct and forged values. This step should not involve a great deal of effort; for example, in the speech example transformation given above, the cognitive process verifying the speaker's voice happens subconsciously.

The human must also be able to compute $D(m,k)$. The difficulty of this step depends on the exact encryption function used. The function must be simple to compute. Ideally it should have low key storage requirements, but in some cases it may be acceptable for the user to carry additional storage (e.g., a piece of paper). In practice, this is probably the step that requires the most effort.

It is possible to reduce the total effort required and increase the chances of success by using a non-keyed transformation or null encryption. However, in any application of this model, the balance of usability versus the security properties listed in section 5 should be taken into account.

The result of the KHAP protocol is that the human participant can verify whether data was modified in transit. However, it specifies no behavior for *H* when a disruption is detected. This behavior will largely be dependent on the problem domain, but some general solutions are mentioned here.

Out of Band Channels It may be possible for *H* to contact *S* using another channel. For example, if a user detects a forgery at an ATM, he may be able to contact the bank by telephone.

Retry Elsewhere In some instances, the effects of a modified message may be nullified by a subsequent message. For example, consider an Internet voting service where only the last vote that has been cast is counted. If a user detects a forgery at one machine, he may try again later from a different location, replacing any potentially forged vote.

Single-bit Communication In some cases, it may be possible to send a single bit of information to the remote system. For example, consider a smartcard system for signing documents. A user types in a document and sends it to the smartcard for signing. The smartcard computes the signature but doesn't transmit it. Instead, it transmits a KHAP-encoded confirmation to the user and waits for a pre-determined timeout period (e.g., 60 seconds). At the end of the timeout period, the card releases the signature. If the user detects a forgery, he removes the smartcard from the reader before the timeout period has expired (if he receives no response at all, he assumes the message was not received and removes the card). Thus the bit is set if the card is left in and unset if the card is removed. The single bit translates to commands of "sign the document" or "don't sign the document" to the card.

7 Use Cases

Because the exact transformations and encryption used depend somewhat on the problem domain, it is useful to examine some specific applications. This section describes some potential sequences of events. The exact details of the transformations in each scenario are meant to show the wide variety of possibilities.

7.1 Electronic Voting

Alice wants to vote electronically at a public Internet terminal. She has a smartcard which will sign her vote, and both she and the card know that their shared key is “cows”. Alice plugs her smartcard into the terminal and makes two votes.

The first vote is a referendum, and she votes “yes.” The terminal sends “1 yes” to the smartcard, which cryptographically signs her vote and sends it anonymously to the vote tallier. The smartcard returns a three dimensional rendering of a group of cows in a field spelling out the string “1 yes”. Alice reads the text, verifies that it matches her vote, and verifies that cows are involved.

The second vote is between several candidates. Alice produces a write-in vote of “Mickey Mouse.” The terminal maliciously sends “2 Donald Duck” to the smartcard, which it signs and sends to the vote tallier. The terminal receives back a rendering of “2 Donald Duck“ branded onto a cow. To try to fool Alice, the terminal renders her original vote, “Mickey Mouse” next to a picture of some sheep. Alice verifies that the vote matches her intent, but is alarmed that there is no cow in the picture.

Alice removes her card from the machine. Later that day, she is at a different public terminal. She inserts her card and recasts her second vote. The voting authority receives her vote and replaces the old, malicious vote with her new vote. She may also anonymously telephone the voting authority to lodge a complaint with the original terminal.

7.2 Document Signing

Recall the example from the beginning of the paper. Alice wants to send a signed document to Bob using her smartcard, but only public conference terminals are available. Fortunately, Alice is presenting a paper on KHAP, and so she has her key ready. Alice's key is a set of speech patterns that were randomly generated; when text is synthesized using the patterns, the resultant voice is easily recognizable to Alice, who has heard it several times before (when using her smartcard). Note that Alice could not easily describe the characteristics of the voice. Fortunately, she doesn't need to; she need only distinguish the voice from other voices.

Alice types the memo into the terminal's word processor program and requests that it be sent to her card for signing. The card simultaneously signs the data and produces a synthesized version of the text. It does not release the signature from the card, but returns an audio file of the synthesized voice. The card then waits for the length of the audio stream plus thirty seconds. If it has received no input by that point, it releases the signature. If the card is removed from the reader, it obviously cannot release the signature.

Alice listens to the voice reading her memo, confirms that it is the voice she recognizes and the text is what she wrote, and waits ten seconds (a timeout period she has pre-determined with the card) for the signature.

8 Future Work

There are several issues with KHAP that have yet to be addressed.

Because of the nature of hard AI problems, there is no way to formally prove that they will remain hard; it is merely the consensus of the AI research community that there is no solution for a particular problem. It is therefore critical to further investigate

specific implementations of KHAP.

It may be possible in some circumstances to gain some knowledge of a specific transformation key over a large number of messages. For example, imagine a system that uses messages encoded as speech with a particular voice key. If an attacker starts with a guess as to the key and refines his estimate as more messages are heard, eventually the difference between his estimate and the key will drop below ε, making the two indistinguishable. One possible way to counter this would be to "age" the key; that is, to change it over time such that the rate of change is greater than the rate at which an attacker's estimate improves. If the changes are small enough, then a user will not be able to distinguish them over a small set of changes. Over a larger set of changes, the user will be incrementally "retrained" from message to message.

One implementation obstacle is the computational power required to perform the transformations. In particular, is it feasible to have a smartcard synthesizing speech or rendering 3D models? Where are the models stored? A possible solution to this is to use external computation and storage. A smartcard may be able to use encrypted auxiliary storage (Maheshwari and Vingralek 2000) to hold the rendering models or the data required for speech synthesis. If a remote coprocessor is available, the smartcard may be able to offload some of the computation (Yee 1994).

9 Conclusions

The use of both malicious terminals and hard AI problems in security are relatively new areas of research brought about by the recent ubiquity of networked computing.

We have introduced KHAP, a system for using hard AI problems to provide data integrity from a malicious terminal in some common circumstances. We have shown that given a sufficiently

hard AI problem, using KHAP succeeds in securing data integrity between a human and a computing system. We have also shown that KHAP-based protocols can be feasibly performed by humans.

Furthermore, we have introduced the concept of keyed AI transformations, a security primitive that can be used in a variety of other protocols both to hide data from automated adversaries and to attach authentication to specific messages. Previous work has demonstrated the use of hard AI problems as security primitives (von Ahn, Blum, Hopper, and Langford 2003); however, we believe this to be the first use of AI problems for message authentication.

Referências

Berta, Istvn Zsolt and Istvn Vajda (2003). Documents from malicious terminals. In *SPIE Microtechnologies for the New Millenium 2003, Bioengineered and Bioinspired Systems, Maspalomas*.

Gobioff, H., S. Smith, J. Tygar, and B. Yee (1996). Smartcards in hostile environments. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*.

Hopper, Nicholas J. and Manuel Blum (2001). Secure human identification protocols. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pp. 52–66. Springer-Verlag.

Kochanski, G., D. Lopresti, and C Shih (2002, Setembro). A reverse turing test using speech. In *Proceedings of the Seventh International Conference on Spoken Language Processing*, pp. 1357–1360. Denver, Colorado.

Maheshwari, Umesh and Radek Vingralek (2000, Outubro).

How to build a trusted database system on untrusted storage. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, pp. 135–150. San Diego.

Matsumoto, T. and H. Imai (1991). Human identification through insecure channel. In D. W. Davies (Ed.), *Advances in Cryptology – EUROCRYPT 91*, Volume 547 of *Lecture Notes in Computer Science*, pp. 409–421. Springer-Verlag.

Monrose, Fabian, Michael K. Reiter, Qi Li, and Susanne Wetzel (2001, Maio). Cryptographic key generation from voice. In *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, California.

Naor, Moni and Benny Pinkas (1997). Visual authentication and identification. In *Advances in Cryptology – Crypto '97*, Volume 1294 of *Lecture Notes in Computer Science*, pp. 322–336. Springer-Verlag.

von Ahn, Luis, Manuel Blum, Nicholas J. Hopper, and John Langford (2003). CAPTCHA: Using hard AI problems for security. In *Advances in Cryptology – Eurocrypt 2003*, Volume 2656 of *Lecture Notes in Computer Science*. Springer-Verlag.

Yee, B. S. (1994). *Using Secure Coprocessors*. Ph.D. thesis, Carnegie Mellon University.