One-Time-Password-Authenticated Key Exchange

Kenneth G. Paterson[1] and Douglas Stebila[2]

[1] Information Security Group
Royal Holloway, University of London, Egham, Surrey, UK

[2] Information Security Institute
Queensland University of Technology, Brisbane, Australia

ACISP 2010 – July 6, 2010

One-Time-Password-Authenticated Key Exchange

Kenneth G. Paterson[1] and Douglas Stebila[2]

[1] Information Security Group
Royal Holloway, University of London, Egham, Surrey, UK

[2] Information Security Institute
Queensland University of Technology, Brisbane, Australia

ACISP 2010 – July 6, 2010

Threats against passwords

1. Random guessing

- Many people use weak passwords and an attacker can guess these in bulk in sufficiently large quantities.
- Imperva (2010)[1] reported a password breach of 32 million accounts on RockYou.com.
- Further analysis of the breached password data revealed:
  - total entropy of passwords: 21.1 bits
  - the top 100 passwords cover 4.6% of accounts
  - the top 1,000 passwords cover 11.3% of accounts
  - the top 10,000 passwords cover 22.3% of accounts
- For user-generated passwords, attackers can break into a random account in a single guess with probability around 2^{-13}.

[1] http://www.imperva.com/docs/WP_Consumer_Password_Worst_Practices.pdf

Threats against passwords

2. Spyware

- Spyware-compromised computers can easily record usernames and passwords and send them to criminals.

3. Phishing

- Many users are tricked into entering their password into the wrong webpage – by clicking on links in email or misunderstanding browser security information.
- "FBI Director Nearly Hooked in Phishing Scam, Swears Off Online Banking" – October 2009[2]

[2] http://www.eweek.com/c/a/Security/FBI-Director-Nearly-Hooked-in-Phishing-Scam-Swears-Off-Online-Banking-616671/

One-time passwords

A system involving a set of passwords, each of which is to be used only once. Passwords could be random, pseudorandom, time-dependent, or challenge/response.

1. **Random guessing**: One-time passwords offer some protection: they are uniformly distributed, but have low entropy $\log_2 10^6 = 20$ bits.

2. **Spyware**: One-time passwords are a good defence: stolen passwords can't be used again.

3. **Phishing**: One-time passwords are still vulnerable: since the stolen password was never used to begin with, it can still be used; e.g., man-in-the-middle attack.

One-time passwords

A system involving a set of passwords, each of which is to be used only once. Passwords could be random, pseudorandom, time-dependent, or challenge/response.

1. **Random guessing**: One-time passwords offer some protection: they are uniformly distributed, but have low entropy $\log_2 10^6 = 20$ bits.

2. **Spyware**: One-time passwords are a good defence: stolen passwords can't be used again.

3. **Phishing**: One-time passwords are still vulnerable: since the stolen password was never used to begin with, it can still be used; e.g., man-in-the-middle attack.

How to deploy one-time passwords to users?

One-time password tokens

Our work

1. How should we model the security of one-time password schemes?
2. Are existing one-time password schemes secure?
3. How should we build secure one-time password schemes?

We show how to use techniques from **password-authenticated key exchange** to further protect one-time passwords from phishing attacks and provide mutual authentication. Our security model allows for pseudorandom and time-dependent passwords.

1. How should we model the security of one-time password schemes?

Basic security goals

- User-to-server authentication based on knowledge of password.
- Secure even if previous passwords revealed.
- Easy to use; hard to screw up.
- Secure even if future passwords revealed.

Basic security goals

- User-to-server authentication based on knowledge of password.
- Secure even if previous passwords revealed.
- Easy to use; hard to screw up.
- Secure even if future passwords revealed.

Additional security goals

- Server-to-user authentication based on knowledge of password.
- Authentication protocol secure against offline dictionary attacks.
  - **Offline dictionary attack:** Observing one protocol run allows an attacker to go through a dictionary of passwords to check if they match the protocol transcript.
- Establishment of secure communications channel.

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

SSL + basic passwords:

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

SSL + basic passwords:

Alice	Bob

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

SSL + basic passwords:

Alice		Bob
	$\xleftarrow{\text{key exchange}} \xrightarrow{}$	

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

SSL + basic passwords:

Alice		Bob
	$\xleftarrow{\text{key exchange}}\xrightarrow{}$	
	$\downarrow$ session key	

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

SSL + basic passwords:

Alice		Bob
	key exchange $\longleftarrow\!\!\longrightarrow$	
	$\downarrow$ session key	
	$\boxed{\text{SSL}}$	

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

SSL + basic passwords:

Alice		Bob
	$\xleftarrow{\text{key exchange}}\ \longrightarrow$	
	$\downarrow$ session key	
password $\longrightarrow$	$\boxed{\text{SSL}}$	

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

SSL + basic passwords:

Alice		Bob

$\xleftarrow{\text{key exchange}}\ \xrightarrow{}$

$\downarrow$ session key

password $\longrightarrow$ $\boxed{\text{SSL}}$ $\longrightarrow$ password? ✓ ✗

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

Password-authenticated key exchange:

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

Password-authenticated key exchange:

Alice	Bob
password	password

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

Password-authenticated key exchange:

Alice	Bob
password	password

$$\xleftarrow{\text{pw. auth}} \xrightarrow{\text{\& key ex.}}$$

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

Password-authenticated key exchange:

Alice		Bob
password		password
	$\xleftrightarrow{\text{pw. auth \& key ex.}}$	
	$\downarrow$ session key	

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

Password-authenticated key exchange:

Alice		Bob
password		password
	$\xleftarrow{\text{pw. auth}}\ \xrightarrow{\text{\& key ex.}}$	
$\swarrow$		$\searrow$
auth? ✓ ×	$\downarrow$ session key	auth? ✓ ×

Background: password-authenticated key exchange

Server and client prove to each other that they know the password without disclosing any useful information about the password; they also get a shared secret out at the end.

Password-authenticated key exchange:

Alice		Bob
password		password
	$\xleftrightarrow{\text{pw. auth \& key ex.}}$	
$\swarrow$		$\searrow$
auth? ✓ ×	$\downarrow$ session key	auth? ✓ ×

Introduced by Bellovin & Merritt (1992); lots of research since then.

Formally modelling security

To show a protocol secure, we:

1. Model the powers of an adversary.
2. Define a game that the adversary has to win in order to break security.
3. Show upper bounds on the probability that an adversary can win the game (possibly related to hard computational problems).

Formal security arguments ("security proofs", "provable security") do not always mean a protocol is secure in practice. But they can still be a good heuristic that the design of the protocol is sound.

Security model

We define a security model for one-time password-authenticated key exchange based on the Bellare-Pointcheval-Rogaway model for PAKE.

The adversary has complete control of the communication links and can direct participants to perform certain actions.

The adversary can:

- modify, reorder, or delete protocol messages
- send protocol messages
- direct participants to perform certain actions
- compromise certain secrets
  - session keys
  - **one-time passwords**

Security goals of the adversary

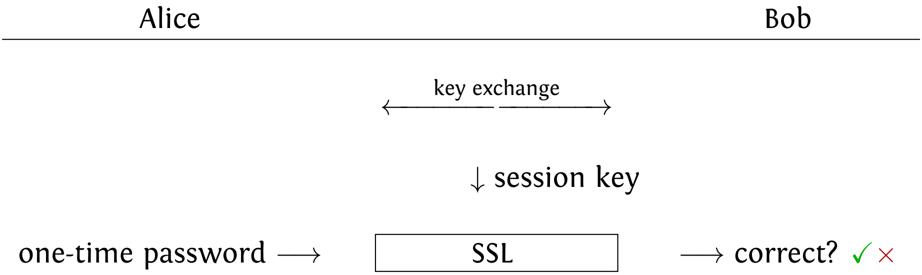
The adversary has two goals:

1. Break confidentiality:
   determine the session key of any "fresh" session.
2. Break authentication:
   impersonate one party in any "fresh" session.

"Fresh": the adversary hasn't revealed the one-time password or session key for that session.

2. Are existing one-time password schemes secure?

SSL + one-time passwords



- **Secure** against passive adversaries and dictionary attacks.
- **Insecure** if SSL server certificate authentication fails.
- **Insecure** if SSL is bypassed.
- Doesn't provide mutual authentication.

OPKeyX protocol

- Proposed by Abdalla, Chevassut, Pointcheval at PKC 2005.
- Uses a hash chain to derive one-time passwords from a seed.
- Server only stores a verifier so it can't impersonate the user.

Alice		Bob
enter pw; select n $V \leftarrow H(H^n(pw))$	$\xrightarrow{V}$	
enter pw; $x \leftarrow H^n(pw)$ $V \leftarrow H(x)$	$\boxed{\text{protocol using } V, x}$	
$\longrightarrow$ ✓ ✗		$\longrightarrow$ ✓ ✗
$n \leftarrow n - 1$		if ✓: $V \leftarrow x$

OPKeyX protocol

- **Secure** against passive adversaries and dictionary attacks.
- **Secure** if passwords are revealed in order.
- **Insecure** if attacker gets a later password.

Not a huge flaw, but requires users to be a bit careful with their passwords and could be exploited by a tricky social engineering attacker.

3. How should we build secure one-time password schemes?

Building secure schemes

Simple answer: Password-authenticated key exchange protocols are also good for protecting one-time passwords.

Complex answer: We describe a generic protocol $1(P)$ for building a secure one-time-password-authenticated key exchange protocol from any password-authenticated key exchange protocol P that preserves security.

Building secure schemes

Protocol $1(P)$ – **Login Phase**			
	Client $\hat{C}$		Server $\hat{S}$
1.		$\xrightarrow{\text{“hello”},\hat{C}}$	
2.			pick $\mathsf{ch} \in \mathsf{Indices}$ s.t. $\mathsf{used}_{\hat{S}}(\hat{C}, \mathsf{ch}) = \mathsf{false}$
3.			$\mathsf{used}_{\hat{S}}(\hat{C}, \mathsf{ch}) \leftarrow \mathsf{true}$
4.		$\Pi^{\hat{C}}_{(\hat{S},\mathsf{ch})} \xleftarrow{\text{“hello”}}$	
5.	if $(\mathsf{used}_{\hat{C}}(\hat{S}, \mathsf{ch}) = \mathsf{true})$ then reject		
6.	$\mathsf{used}_{\hat{C}}(\hat{S}, \mathsf{ch}) \leftarrow \mathsf{true}$		
7.	run protocol P with users $(\hat{C}, \hat{S}, \mathsf{ch})$ and $(\hat{S}, \hat{C}, \mathsf{ch})$ and password $\mathsf{pw}_{(\hat{C},\hat{S},\mathsf{ch}),(\hat{S},\hat{C},\mathsf{ch})}$		
8.	if P accepts then		if P accepts then
8.a)	$\mathsf{sid}_{1(P)} \leftarrow \mathsf{sid}_P$; $\mathsf{pid} \leftarrow \hat{S}$		$\mathsf{sid}_{1(P)} \leftarrow \mathsf{sid}_P$; $\mathsf{pid} \leftarrow \hat{C}$
8.b)	$\mathsf{sk}_{1(P)} \leftarrow \mathsf{sk}_P$		$\mathsf{sk}_{1(P)} \leftarrow \mathsf{sk}_P$
8.c)	accept in $1(P)$		accept in $1(P)$
9.	if P terminates then terminate		if P termiantes then terminate
10.	if P rejects then reject		if P rejects then reject

Building secure schemes

Isn't this overkill? PAKE protocols are pretty heavy weight...

Building secure schemes

Isn't this overkill? PAKE protocols are pretty heavy weight...

The heavy lifting in most PAKE protocols – such as Diffie-Hellman – is used to protect against offline dictionary attacks. If we want this protection for one-time password protocols, then there may not be much more we can optimize.

Pseudorandom passwords

The main proof assumes perfectly random passwords, but using pseudorandomly generated passwords from a single seed is fine (assuming a good pseudorandom number generator F).

$$pw_1 = F(seed, 1), pw_2 = F(seed, 2), \ldots$$

If the client and server get out of sync on which is the current password, they'll never get back in sync.

Pseudorandom passwords

The main proof assumes perfectly random passwords, but using pseudorandomly generated passwords from a single seed is fine (assuming a good pseudorandom number generator F).

$$pw_1 = F(seed, 1), pw_2 = F(seed, 2), \ldots$$

If the client and server get out of sync on which is the current password, they'll never get back in sync.

Challenge/response pseudorandom passwords:

$$pw = F(seed, challenge)$$

This requires communication of the challenge, adding an extra round of communication. Need to ensure challenges aren't reused.

Time-dependent pseudorandom passwords

To use pseudorandom passwords without additional communication, try time-dependent passwords:

$$pw = F(seed, t)$$

where t is the device's current time.

But then we have to deal with time synchronization.

- Network time servers are inconvenient (require trust and communication).
- Can't allow the server to accept a few recent passwords as the server has to "commit" to one single password in the protocol.
- Solution: just send the time in cleartext and let the server decide if it's acceptable or not.

Summary

One-time password systems are being deployed by banks and businesses to reduce the damage of spyware attacks.

We have shown how to use techniques from **password-authenticated key exchange** to further protect one-time passwords from phishing attacks and provide mutual authentication. Our security model allows for pseudorandom and time-dependent passwords.

May be useful in online banking to protect against phishing attacks. More immediate and easier-to-deploy application: VPN and corporate webmail.

Open question: Can you do secure one-time-password authentication without heavy weight public key constructions? (Conjecture: no.)