

Off-the-Record Messaging

Or, When Not To Use PGP

Black Hat Briefings
31 July, 2002

Ian Goldberg

Zero-Knowledge Systems
ian@zeroknowledge.com

Joint work with Nikita Borisov and Eric Brewer, UC Berkeley

Outline

- Introduction
- Motivation
- Communication using PGP
 - Public-key encryption
 - Digital signatures
- Problems with PGP
- Towards a better solution
 - "Off-the-record" Messaging
- Different cryptographic primitives
 - Perfect forward secrecy
 - Message authentication codes
 - Malleable encryption
- Putting it together
- Conclusions

Introduction

- Alice and Bob wish to have a secure private conversation.
(What do we mean by this?)
- We'll start with what most people mean:
 - No one other than Alice and Bob can read the messages sent between them.
 - When Bob receives a message from Alice, he can be assured that Alice was the one that sent it, and it hasn't been modified in transit.
 - ▷ And vice versa, of course
- Alice and Bob know a thing or two about cryptography, so they get a copy of PGP (or gnupg, or whatever), and start exchanging messages.
- We'll even be generous and assume they have some secure method of validating each other's public keys.

Motivation

- PGP in fact satisfies the stated goals, so what's the problem?
- The problem is that the criteria we've listed so far for a "secure private conversation" don't really match what we intuitively mean.
- Alice and Bob are still vulnerable to attacks that shouldn't exist if the conversation were truly secure and private.
- Let's see how.

Communicating using PGP

Quick review of public-key concepts in PGP:

- Public-key crypto is like a lockbox that has separate keys for locking and for unlocking.
 - You can't lock with the unlocking key, or unlock with the locking key.
 - We (the scientific community) figured out how to do this with math instead of with metal in the late 1970's.

- Anyone can generate brand-new pairs of locking and unlocking keys. (Called "keypairs".)
 - But if you only have one of the keys, you can't figure out the other.

- In PGP, the user makes two such keypairs:
 - One for public-key encryption
 - One for digital signatures

Public-key encryption in PGP

□ Bob's setup

- Bob makes a keypair (known as the "encryption keypair").
- He widely publishes copies of the encryption locking key, but keeps the unlocking key to himself.

□ Alice wants to send a message to Bob

- Alice gets a copy of Bob's locking key, and uses it to lock her message.
- She then sends the message to Bob over a public channel.

□ Eve wants to read Alice's message to Bob

- Eve can get a hold of Bob's locking key (as can everyone), but not his unlocking key.
- Eve can't read the message.
 - Not even Alice can unlock the message she herself locked!

□ Bob can read the message

- He uses his unlocking key.

Digital signatures in PGP

□ Alice's setup

- Alice makes a keypair (known as the "signature keypair").
- She widely publishes the unlocking key, and keeps the locking key to herself.

□ Alice sends a message to Bob

- Alice uses her signature locking key to lock the message.
- She then sends the resulting locked message to Bob.

□ Anyone can ascertain that Alice was the one that sent the message

- They just try to use Alice's signature unlocking key to unlock the message. If it works, Alice's locking key must have been used to lock it, and Alice is the only one with that key.

□ Also, no one could have modified the message

- Any changes to the message after it's locked will cause the unlocking to fail.

Encrypting and signing in PGP

With PGP, we usually use both encryption and digital signatures:

- Alice takes the message, locks it once with her own signature key, and then locks that result with Bob's encryption key.
- Eve can't unlock the outer box, since she doesn't have Bob's encryption unlocking key
 - though she does have Alice's signature unlocking key, as does everyone
- Bob can unlock the outer box, and then unlock the inner box using Alice's signature unlocking key, and discover the secret message, and be sure that:
 - Alice was the one that sent it
 - It hasn't been modified since Alice sent it

Problems with PGP

So what's the problem? People use this all the time!

- The big problems happen when Bob's computer is owned by Carl the cracker.
- Carl can recover Bob's encryption unlocking key. What does this let him do?
 - If Bob doesn't realize this has happened, Carl will certainly be able to read future messages from Alice to Bob.
 - If Bob does realize it, he can change his encryption keypair.
- But: even if Bob changes his encryption keypair, Carl can now read all of Bob's past messages!
- Not only that, but he can prove to others that Alice wrote those messages, since they're digitally signed.

Problems with PGP

PGP doesn't provide two properties that we really want in a system intended for private conversations:

- Perfect forward secrecy

- The assurance that once Bob has received Alice's message and destroyed his copy, it will be impossible for Eve or Carl to decrypt it later on, even if all of Alice and Bob's secrets become compromised.

- Repudiability

- The ability for Alice to deny that she wrote any particular message.
- Digital signatures are often touted as great tools for achieving non-repudiability. That's perfect for things like electronic contracts, but terrible for private communication.
- Note that Alice could just not sign the message, but then not even Bob is assured of the message's authenticity!

"Off-the-record" Messaging

We want a system for online communication that behaves much more like a casual, "off-the-record" conversation. In particular:

- If no one was actively listening in at the time the conversation was taking place, then the conversation's contents are secure forever.
 - Unless of course Alice or Bob divulges the information.
- Even if Bob (or Alice) goes to a third party Trent with the contents of the conversation, Trent will have to take his word for it.
 - There's nothing that can be used to prove the authenticity of the messages to him.
- On the other hand, Alice and Bob are assured of the authenticity of each other's messages.

Different cryptographic primitives

In order to achieve these goals (and in particular the last two, which seem conflicting at first glance), we will need to use cryptographic techniques a bit more esoteric than plain old public-key encryption and signatures:

- Perfect forward secrecy

- Ensuring that key compromise can't be used to recover past messages

- Message authentication codes

- Enabling Alice and Bob to be assured of the authenticity of each others' messages, without Trent being able to

- Malleable encryption

- Allowing anyone to easily modify transcripts of encrypted conversations, to further aid Alice's repudiability

Perfect forward secrecy

- If Carl has compromised either Alice or Bob's computer at the time a message is being composed and sent (or received and read), then he can clearly read the message.

- However, once the compromise is detected and corrected, future messages should not be exposed.
 - This means that encryption key material has to be generated "freshly", and not derived from some stored state.

- In addition, a compromise must never expose past messages.
 - This is what we mean by "perfect forward secrecy".

Perfect forward secrecy

How do we achieve this?

- If a compromise must not expose the contents of past messages, then it must be the case that the encryption keys to those past messages can no longer be calculated from any information on the box.
- In particular, this means that we need to frequently change our encryption keypair.
 - How frequently? As often as is reasonable, taking into account the cost of changing your keys.
 - It turns out it's often not unreasonable to use a different encryption keypair for every message.
 - Note that the keys need to have a large fresh random component.
 - They can't just be a hash of the current time and some master secret, for example.

The costs of perfect forward secrecy

How expensive is it to change keys?

- It depends what cryptosystem you're using for public-key encryption.

- If you're using something like RSA, keypair generation can be pretty expensive.
 - Several seconds of calculation on a modern processor
 - Several minutes on an old PDA

- If you're using something like El Gamal or Diffie-Hellman, it's much cheaper.
 - One modular exponentiation
 - About .1 seconds on a modern processor
 - Still several seconds on an old PDA
 - You might not want to do it every message in this case

The costs of perfect forward secrecy

- Then there's the cost of distributing the new keys, and keep track of your old ones
 - in case a message comes in encrypted with an old key
- Distributing new keys is straightforward
 - Piggyback them on your next message
- Don't keep old keys too long!
 - Otherwise a compromise will expose old data
 - Just keeping the current key and the previous key is usually enough.
 - If a really old message does come in, too bad.
- You may keep a separate encryption keypair (and previous keypair) for each person you're actively communicating with.
 - That way, you can make a new key for a person as soon as you receive a message from that person encrypted with your current key.

Changing your keys

- For each person you're actively communicating with, you need to remember:
 - His current encryption locking key for you
 - Your current encryption keypair for him
 - Your previous encryption keypair for him
- When you send a message:
 - Use his current encryption locking key to lock it
 - Include a copy of your current encryption locking key.
- When you receive a message:
 - If it's locked with your current locking key, generate a new encryption keypair.
 - Forget your previous keypair
 - Make the current keypair your previous keypair
 - Make the new keypair your current keypair.
 - Remember the locking key included in the message as his current encryption locking key for you.

One more detail

We stick with the "usual" thing that's done in public-key cryptography. By "Use his current encryption locking key to lock it", we really mean:

- Use Diffie-Hellman to combine your and his encryption keys to produce a shared secret known only by the two of you.
- Use that shared secret to encrypt the message using AES.
- This is more efficient for longer messages, and the resulting shared secret (which changes every time either of us changes his encryption keypair) will be useful to us later.

Message integrity

Now we turn our attention from confidentiality to integrity. We need to arrange the following situation:

- Bob receives a message, purportedly from Alice.
- Bob is assured that the message actually came from Alice, and has not been modified in transit.
- No one, not even Bob, can assure anyone else of this fact.
 - Anyone else will have to take Bob's word for it.

How can we arrange this?

Message integrity

A useful observation:

- Arrange that messages are cryptographically authenticated using a key known to both Alice and Bob.
 - and no one else
- Then Bob will know that, since he didn't write the message, Alice must have.
- But anyone Bob shows the message to has no such assurance; Bob could have made it up himself.

The cryptographic tool we use for this is called a Message Authentication Code, or MAC.

Message authentication codes

A MAC is to a public-key digital signature what symmetric encryption is to public-key encryption.

- Alice and Bob share a MAC key.
 - This isn't the same as the shared encryption key earlier, but it can be related.
- This key is used both for creating the authenticator ("tag") for a message, as well as for checking it.
 - To check the tag, Bob just re-creates it, and checks that it matches.
- No one except for Alice and Bob can forge authentic-looking messages.
 - They can't even check messages for authenticity just yet, but read on...

Repudiability

This mechanism prevents Bob from convincingly snitching on Alice (or vice versa). But what about Carl?

- Let's say Carl steals Bob's keys, and records a transcript of the encrypted conversation.
- Then he surely could read the communication. There's nothing we can do about that.
 - But let's try to keep him from being able to prove anything about that communication to anyone else!
- Use the same trick as before:
 - Ensure that it's obvious that Carl knows the MAC key. Then no one he shows the transcript to has any reason to believe he didn't make it up himself.
 - Calculate the MAC by taking a simple one-way hash function of the shared encryption key.
 - That way, anyone that can read a message can automatically forge the message.

Repudiability

- Now what about Eve?
 - Eve just records a transcript of the encryption conversation, but can't necessarily read it.

- Most cryptosystems are "non-malleable".
 - This means (roughly) that it's hard to modify an encrypted message so that it still decrypts to something sensible.
 - Assuming you don't know the encryption key, of course.

- If Eve captures a transcript in a non-malleable system, and, separately, Carl comes up with the decryption keys, then it's pretty clear that Eve couldn't have modified the transcript, and the decryption is convincing.
 - So we want to ensure that Eve can alter a transcript even if she doesn't know Bob's secrets!

Malleable encryption

In this case, we want malleable encryption. We want to make it as easy as possible for Eve to sensibly alter transcripts, even if she can't decrypt them.

There are a number of choices for this:

- An "XOR" mode of any block cipher (like AES)
 - OFB mode
 - CTR mode
- Just use any stream cipher
 - RC4, for example

- Each of these works by cryptographically producing a pseudorandom pad (based on a key), and then just XOR'ing the pad with the plaintext to get the ciphertext.
 - To decrypt, just produce the same pad using the same key, and XOR the pad with the ciphertext to recover the plaintext.

Malleable encryption

- To change the transcript, just flip some bits of the ciphertext.
 - The corresponding bits will flip in the plaintext.
- That's as easy as it gets, if you can't read the message!
- If you know part of the message (because of known message structure, headers, etc.), then it's certainly trivial to change that part.

- But wait! What about this MAC we're using? Any change Eve makes to the transcript will cause the MAC to fail, and she'll be detected, right?
 - We need to enable Eve to calculate the correct MAC tag.

What're you, nuts?!

- The whole point of the MAC was so that Bob could be assured that Alice was the one who sent the message, and that it hasn't been modified!

- In order for Eve to calculate the correct MAC for the modified transcript, she'll need the MAC key for the message. Won't that enable her to forge messages?

- Here's the key idea: Bob only needs to be assured of the authenticity of the message at the time he receives it.
 - Once he's received the message, it doesn't matter if Eve can modify it in her transcript.

Publishing MAC keys

- Remember that the MAC key is the hash of the encryption key, and the encryption key changes regularly (roughly every message).
 - So the MAC key also changes regularly.

- Once Bob changes the MAC key (and forgets the corresponding encryption key), he publishes the MAC key by including it in the header of the next message he sends out.
 - It's not part of the encrypted and authenticated message.
 - Therefore, shortly after Bob receives a message from Alice (whose authenticity is cryptographically protected), Eve is trivially able to alter the transcript at will.

Putting it together

We combine these techniques of:

- Perfect forward secrecy
- Message authentication codes
- Malleable encryption

to produce the "off-the-record messaging" protocol.

We have a simple implementation of this protocol as a library, which we've hooked in to the gaim instant messaging client.

- The GUI still sucks, but feel free to improve it. :-)

Conclusions

"Traditional" public-key cryptography like PGP works great when nothing goes wrong.

- But keys can be stolen, or the person you're talking to could be a snitch.

Other cryptographic techniques should be used to mitigate the harm of a compromise.

- Perfect forward secrecy
- Repudiability

More information:

<http://www.cypherpunks.ca/otr/>