Cryptography

digital signatures

digital signatures

Digital signatures

You want to send me an *authenticated* e-mail.

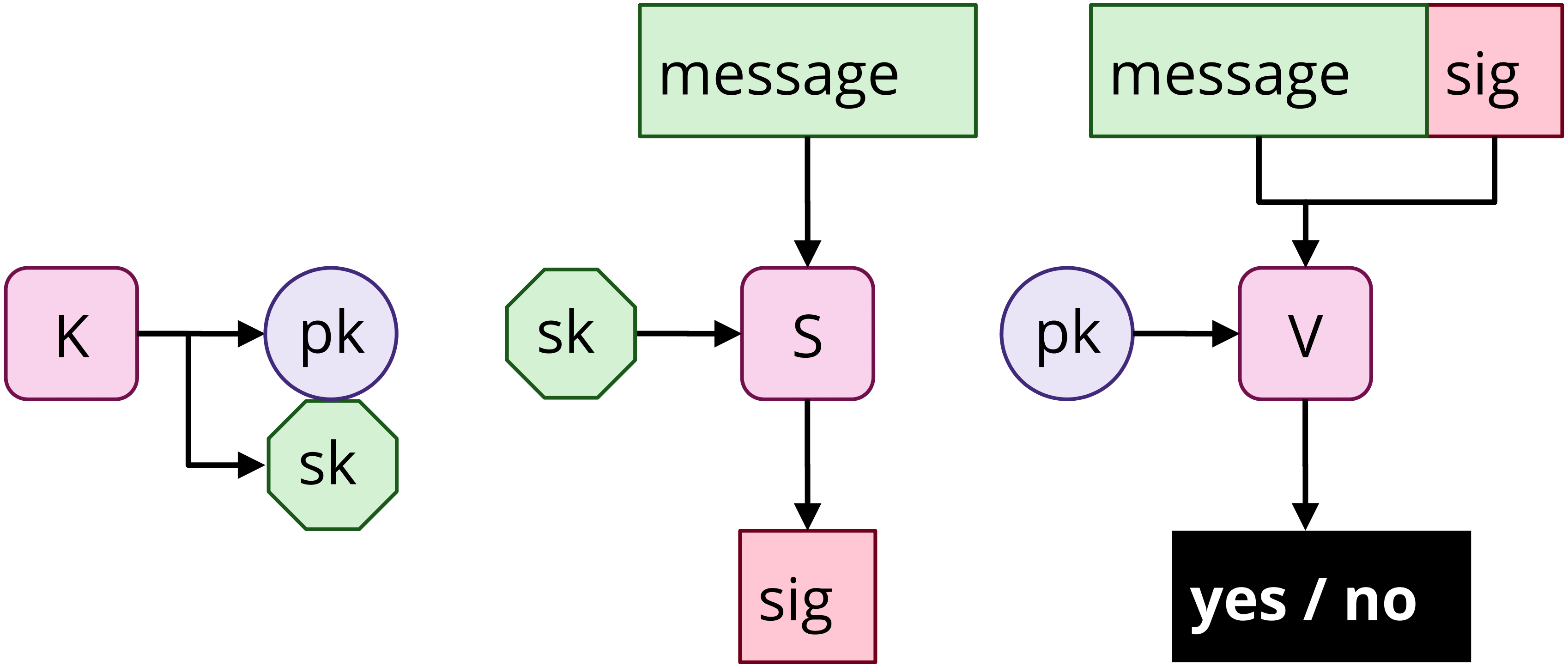
MAC? Possible, we need to share a key.

I (the receiver) can’t prove to anyone else that you sent it – I could have computed the MAC myself.

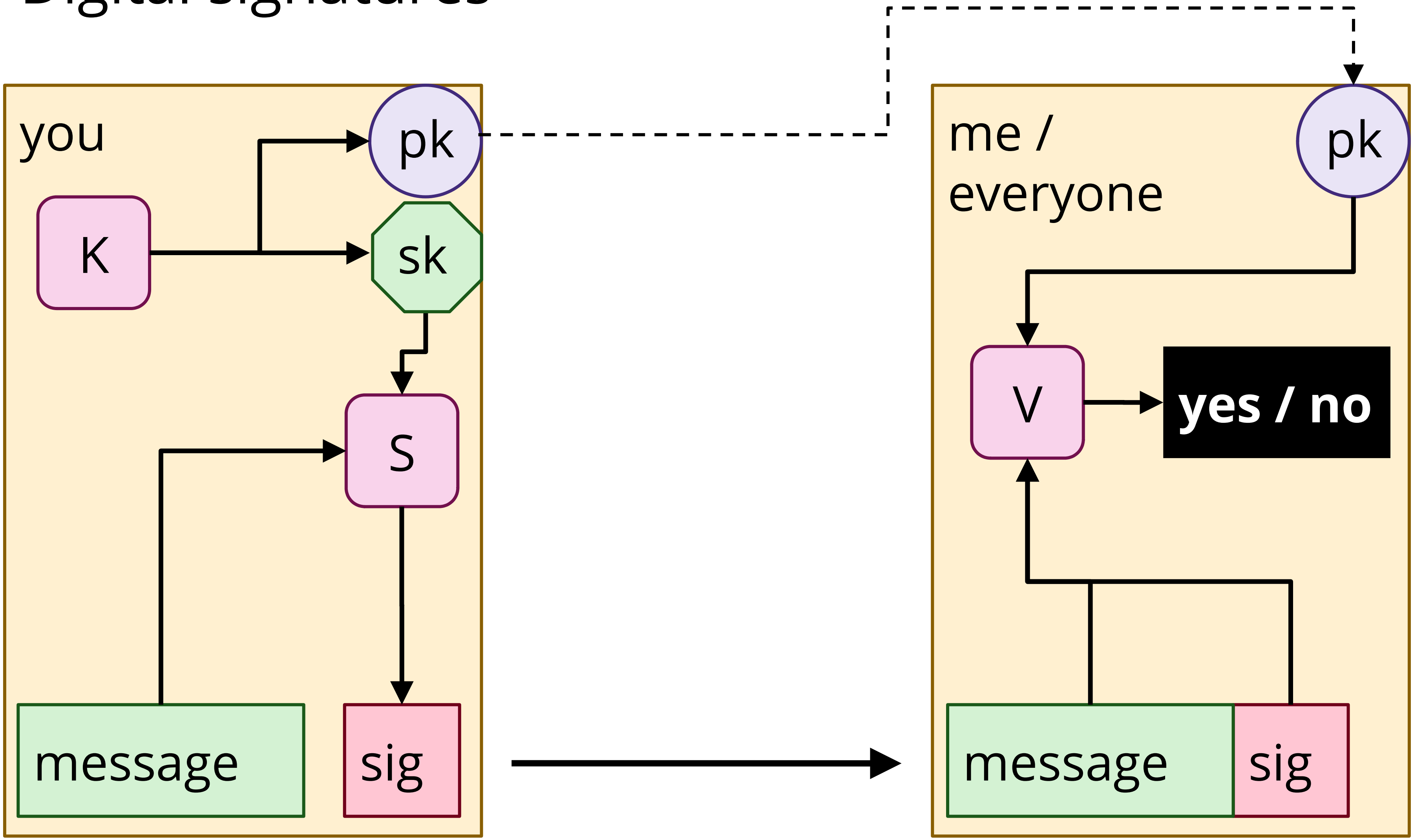
You want to let the public download a copy of your software, and verify that it hasn’t been modified.

MAC is now useless - everyone would need the key!

Digital signatures



Digital signatures



Digital signature scheme

A digital signature scheme contains three algorithms:

The key generation algorithm **K** creates a keypair (pk, sk) containing a public and a secret key.

The signing algorithm **S** takes a secret key and a message and returns a signature.

The verification algorithm **V** takes a public key, a message and a signature and returns “yes” or “no”.

If (pk, sk) comes from K then **V(pk, m, S(sk, m)) = “yes”**.

Notes on digital signatures

The signature itself has a fixed size – however long the message. (Usually one signs a hash of the message.)

A digital signature, unlike a handwritten one, does not change the original document – it’s an extra piece of data.

If you sign two different documents, the two digital signatures will not be the same.

PGP / GPG

PGP / GPG let you sign as well as encrypt messages.

To encrypt a message, you don’t need to enter any password - you only need the recipient’s public key.

To sign a message as well, you need your own secret key, so you’ll be asked the password for it.

What does a digital signature prove?

What does a digital signature prove?

V(pk, m, s) = “yes” means that someone with the sk matching pk has signed m.

If we know that **pk** belongs to a particular person (perhaps it’s hosted on their site or in an official directory), we can infer that this person signed the message…

…or their key has been compromised.

Non-repudiation

If you sign a message, anyone with the signature can prove to a third party *with an authenticated copy of your public key* that you signed the message (or your key has been compromised).

PKI

PKI = public key infrastructure
a.k.a. key management, part 2

Key authentication

From: matt25519@hotmail.com
Subject: copy of document ?

Hi,

Have you still got a copy of secret document D-200-A1X ?
I seem to have lost mine. Can you please encrypt it and
send it to me - my public key is:

5qCJUHixtK2y2BEH5Sk1hzadHMe39GRGEqqHTZRuoQ1hZ

Thanks,
Matt

In public-key cryptography, encrypting a message under a key where you’re not certain that it belongs to the correct person is a security vulnerability.

Key authentication

Public-key cryptography turns a key distribution problem into a key authentication problem.

Sending a document encrypted under a key you just got by e-mail: you have no idea who you’re sending it to, and thus who can decrypt it.

A signature under an unauthenticated key proves nothing (except that the signer knows how to create their own keypair).

How can we get authenticated public keys?

How can we get authenticated public keys?

distribute in person

publish in directory, on personal website …

chain of trust

web of trust

Distribute keys in person

Slow, relatively impractical - but very secure if done well

Imagine that you couldn’t talk to anyone over the internet that you hadn’t met in person before.

Or that you had to visit a company’s offices to pick up keys – and show three different pieces of ID - just so you can visit their website.

We’ll practice exchanging keys in person in the labs!

Publish keys in a directory or a website

This just turns “trust the key” into “trust the directory/site”, but removes the need to meet in person.

A company, university etc. can have its own key directory integrated into its address book system.

Individuals can publish keys in their website, chat / social media accounts, CVs etc.

But if your site, directory or profile is hacked - back to square one.

For the record

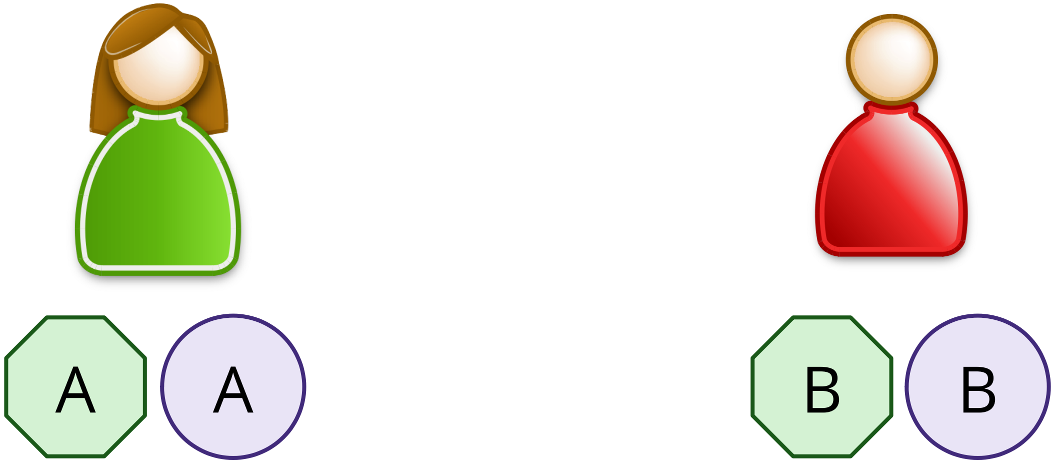
Our PGP public key for this unit has fingerprint

9E8C 9C6F CC53 935F 1BF1 4E01 C205 68FD 3FD7 A820

and is now available on the unit website.

web of trust

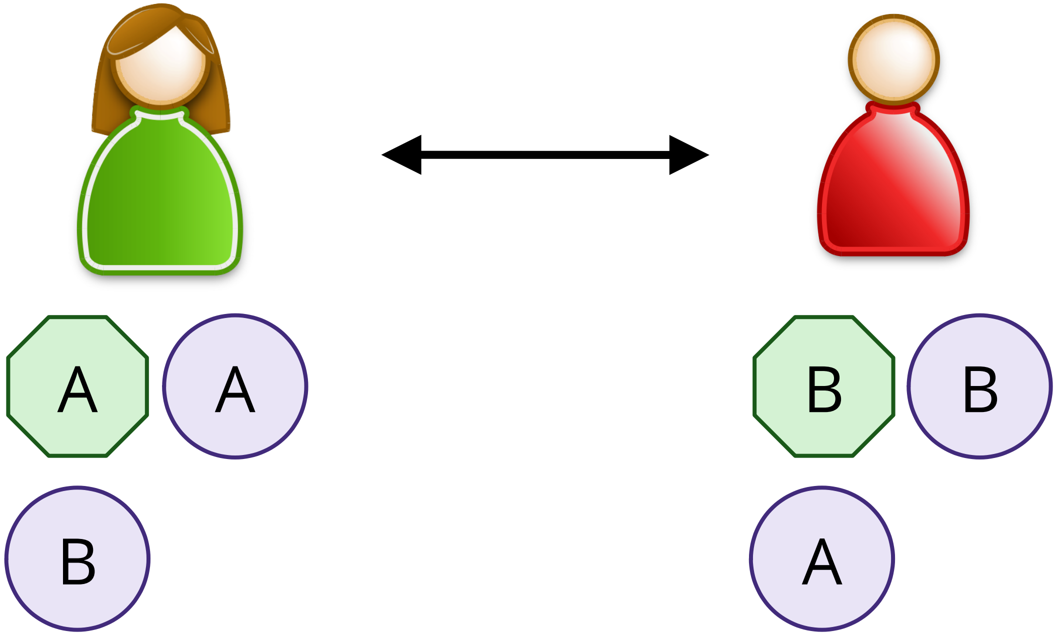
Web of trust



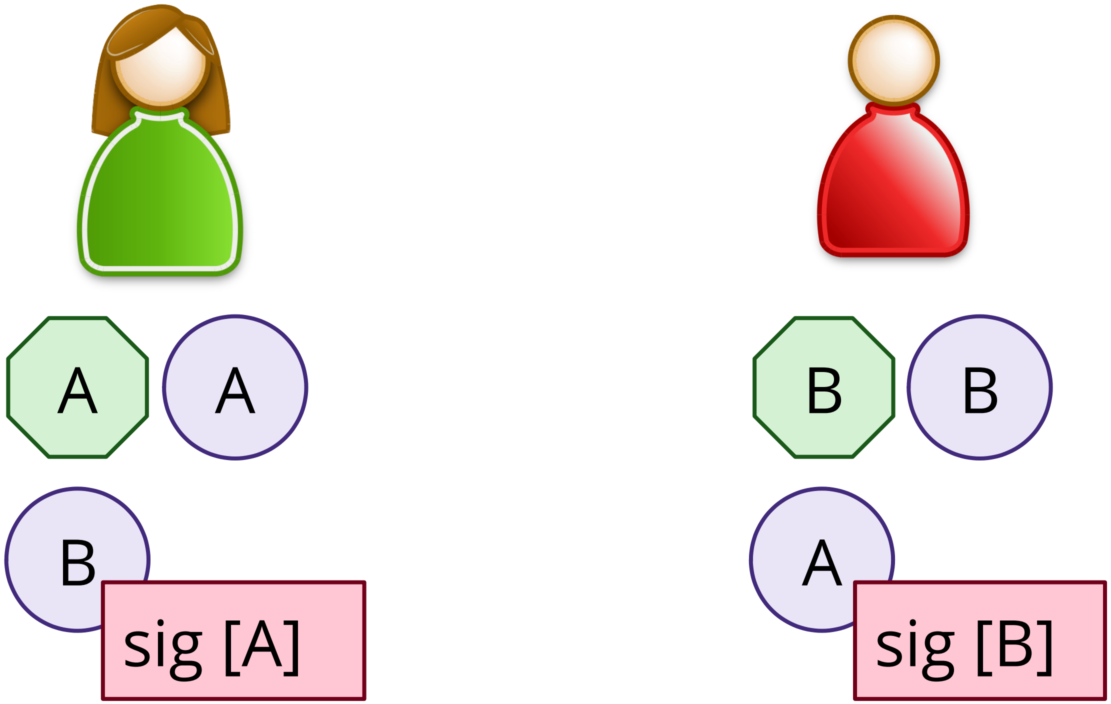
legend: A secret key

A public key

Web of trust



Web of trust



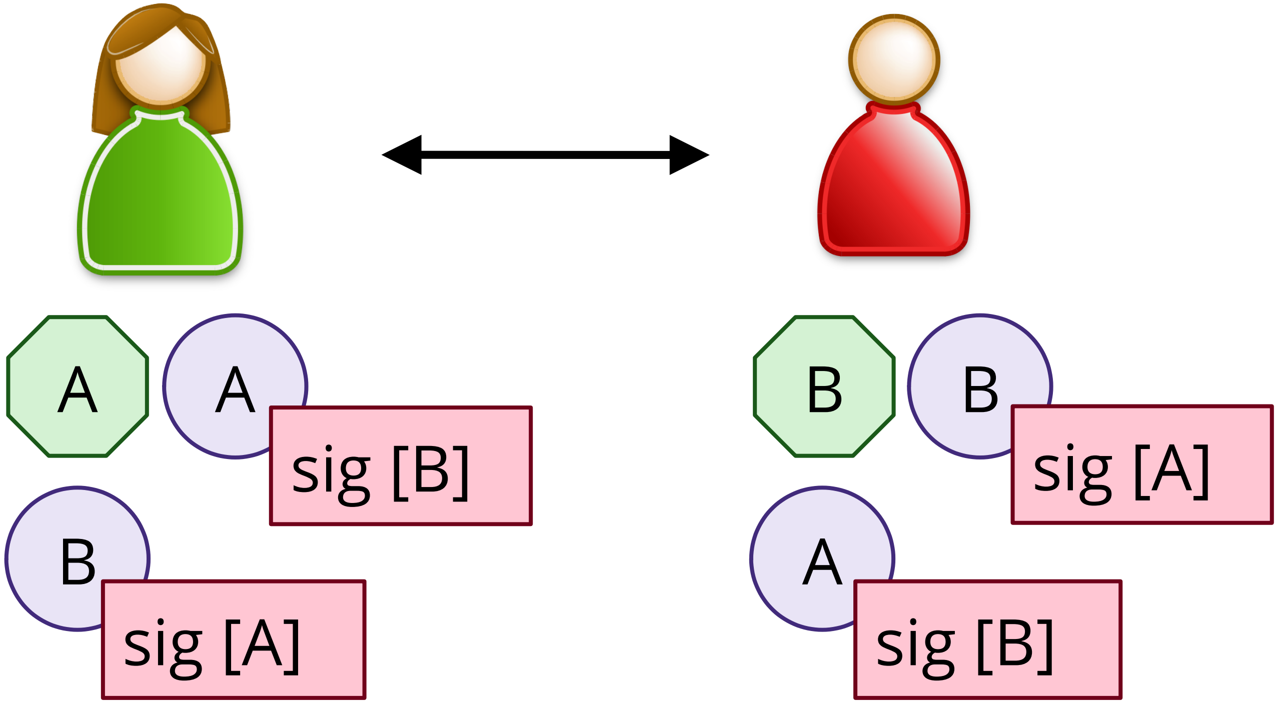
A A

B sig [A]

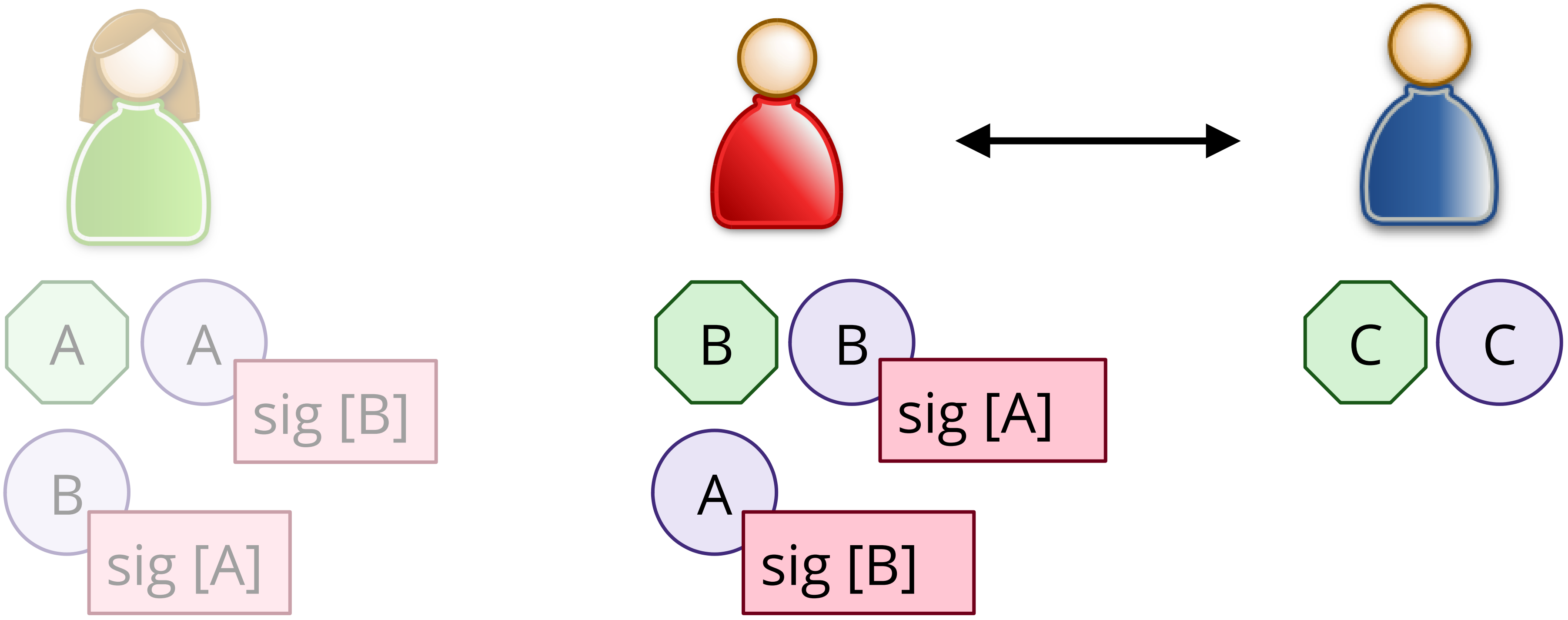
B B

A sig [B]

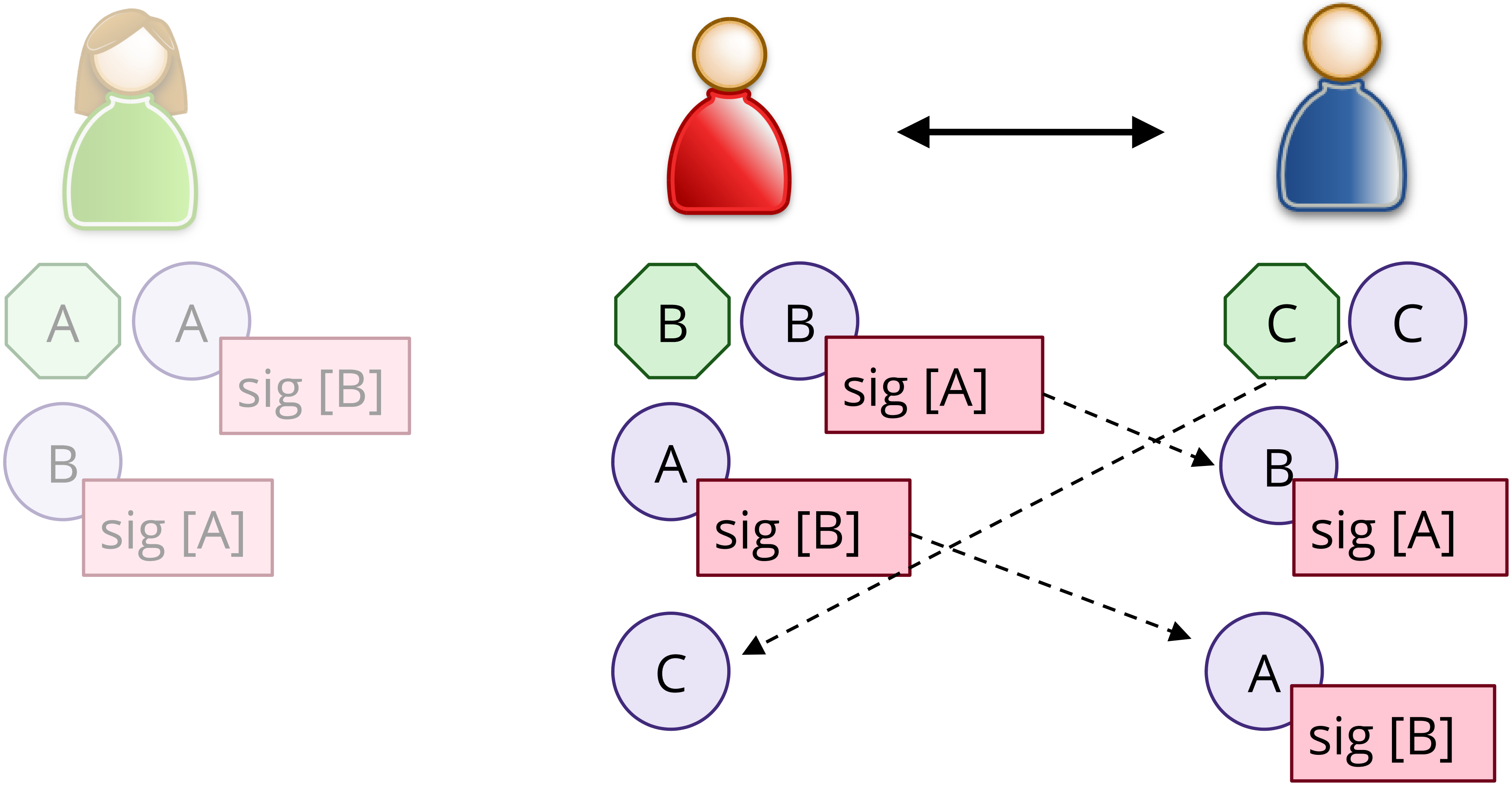
Web of trust



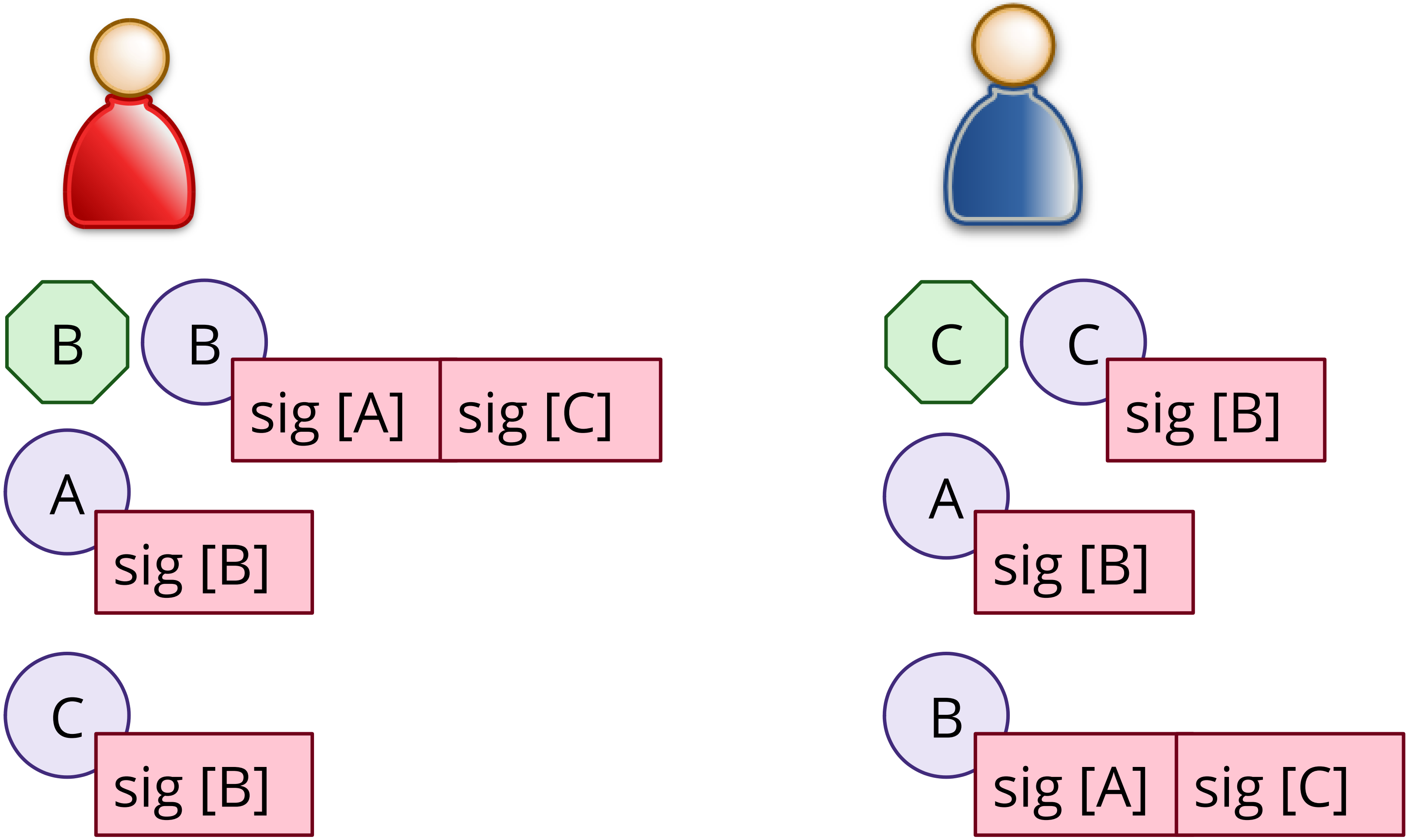
Web of trust



Web of trust



Web of trust



B

B sig [A] sig [C]

A sig [B]

C sig [B]

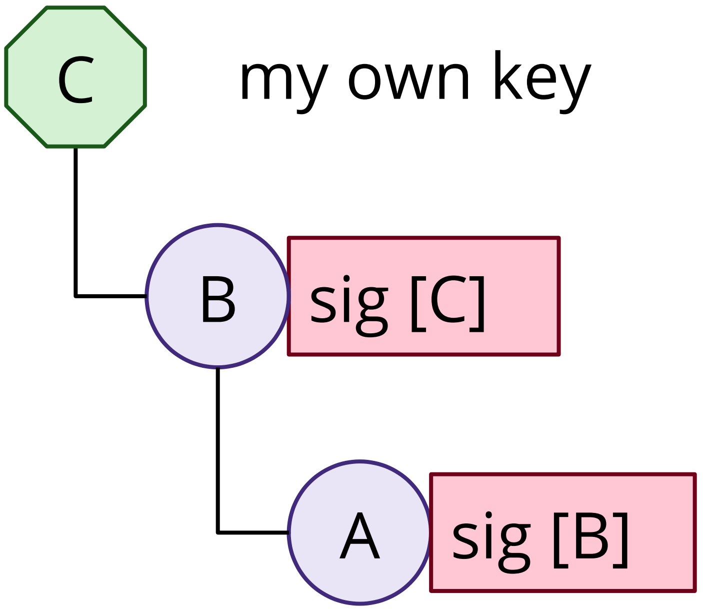
C

C sig [B]

A sig [B]

B sig [A] sig [C]

Chain of trust



C

my own key

B sig [C]

someone I
saw in person

A sig [B]

someone that B
saw in person

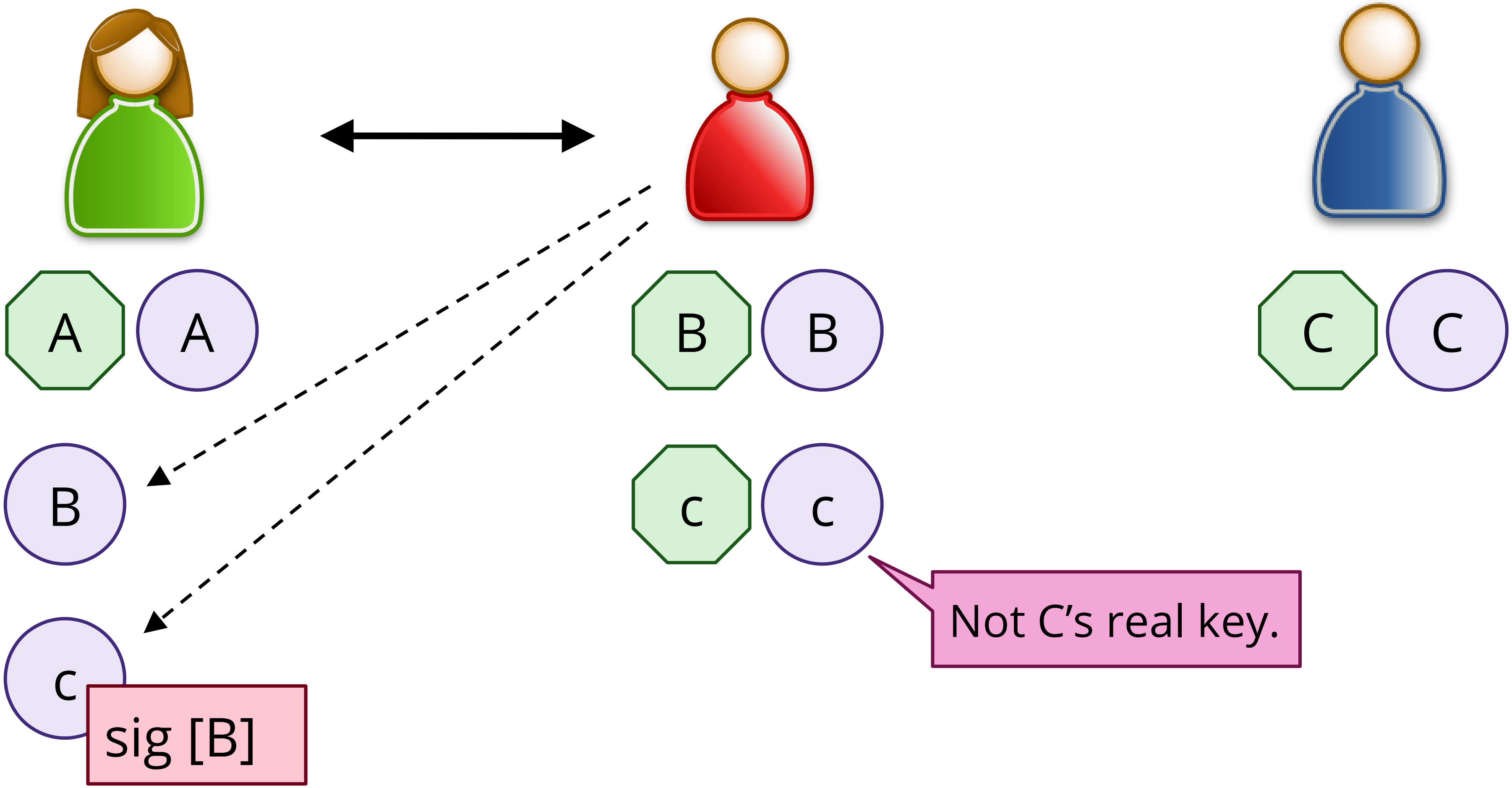
Web of trust - bonus feature

What if someone tries to add their own key to your keyring, under someone else’s name, when you’re not looking?

They can’t sign anything with your key - they don’t have the password.

They can’t include a key without a signature from someone you’ve already got a key from.

Web of trust



Web of trust

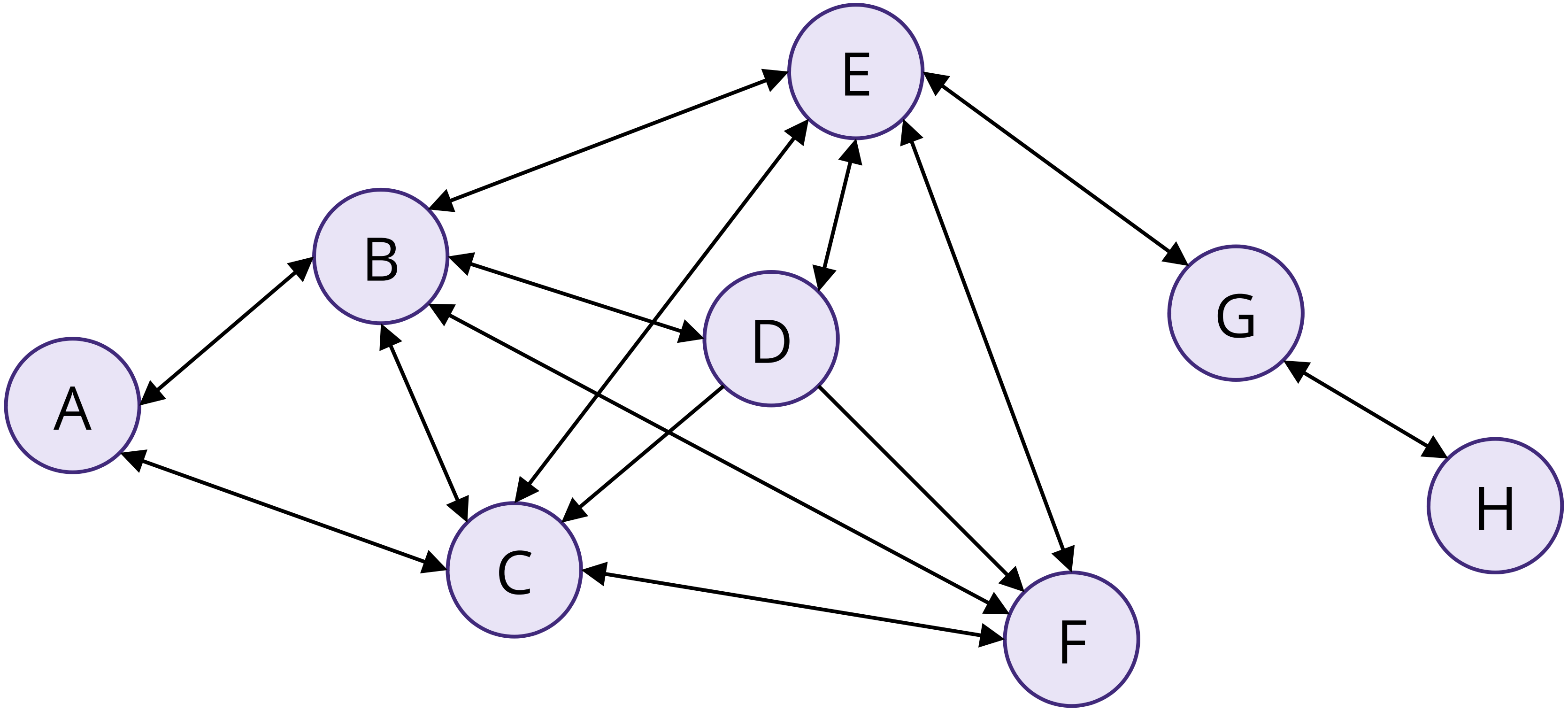
Scenario: B creates a key pair under C's name and claims it came from C. B also signs the "C" public key with B's own key, and sends this to A.

If A now sends an encrypted message to C with this key and B intercepts it, B can read it.

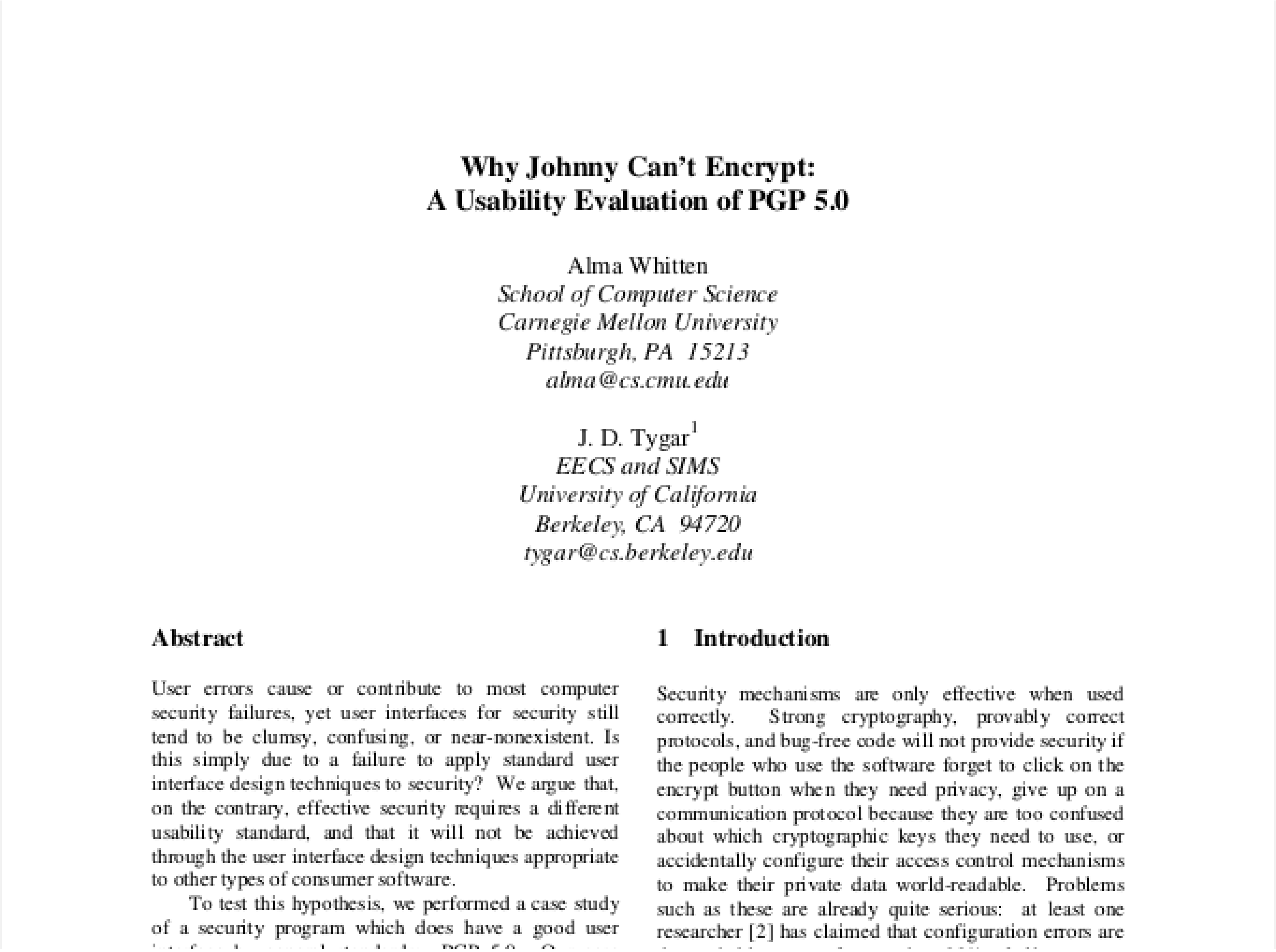
Before accepting a key from someone else on behalf of a third party: do you trust them?

One possible solution: require a third-party key to be signed by multiple people.

Web of trust (theory)



Web of trust (practice)



Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0

Alma Whitten
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
alma@cs.cmu.edu

J. D. Tygar[1]
EECS and SIMS
University of California
Berkeley, CA 94720
tygar@cs.berkeley.edu

Abstract

User errors cause or contribute to most computer security failures, yet user interfaces for security still tend to be clumsy, confusing, or near-nonexistent. Is this simply due to a failure to apply standard user interface design techniques to security? We argue that, on the contrary, effective security requires a different usability standard, and that it will not be achieved through the user interface design techniques appropriate to other types of consumer software.

To test this hypothesis, we performed a case study of a security program which does have a good user

1 Introduction

Security mechanisms are only effective when used correctly. Strong cryptography, provably correct protocols, and bug-free code will not provide security if the people who use the software forget to click on the encrypt button when they need privacy, give up on a communication protocol because they are too confused about which cryptographic keys they need to use, or accidentally configure their access control mechanisms to make their private data world-readable. Problems such as these are already quite serious: at least one researcher [2] has claimed that configuration errors are

Some PGP terminology

ultimate trust = this is my own key

full trust = I trust this person to sign others’ keys

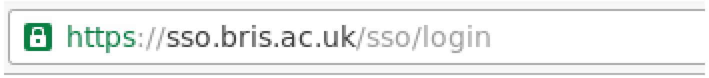
marginal trust = I trust this person’s key, but I don’t completely trust them to sign others’ keys

By default, GPG accepts a key with one full or ultimately trusted signature or at least three marginally trusted ones.

Also, chains of trust can be at most 5 steps long.

Chain of trust

How do I know that this is:



https://sso.bris.ac.uk/sso/login

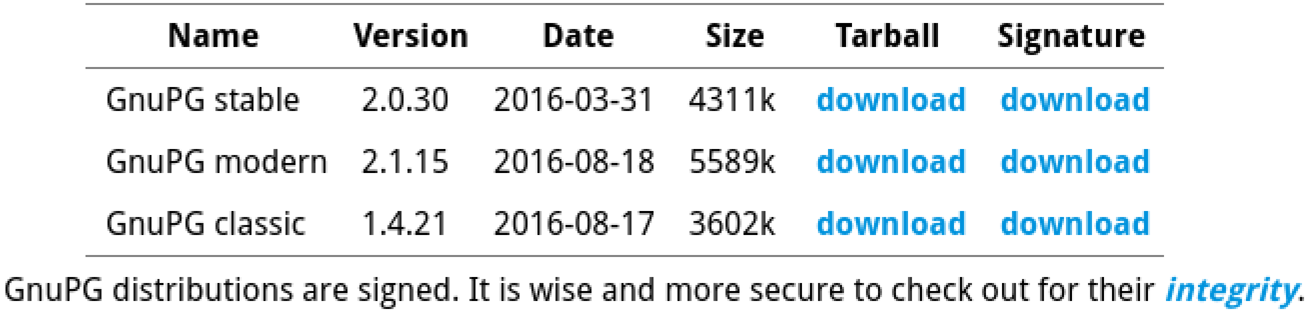
the real UoB ?

gnupg-1.2.tar.gz

the real gnupg ?

when there are so many websites and software packages out there?

Software signatures



Name	Version	Date	Size	Tarball	Signature
GnuPG stable	2.0.30	2016-03-31	4311k	download	download
GnuPG modern	2.1.15	2016-08-18	5589k	download	download
GnuPG classic	1.4.21	2016-08-17	3602k	download	download

GnuPG distributions are signed. It is wise and more secure to check out for their ***integrity***.

gnupg.org download page: along with each file you can download a signature under the author's key.

The idea is that if you have an old version of gnupg, you can verify a new version before you install it.

When you install windows/mac/linux updates, these are signed and the OS verifies them before updating.

Package managers

Windows updates, Linux/Mac OS packages, package managers of programming languages etc …

- master public key distributed with package manager (this belongs to the package manager developers)
- developers of package manager sign keys of individual package developers
- package developers sign their packages with their keys

The package manager installs only packages with a chain of trust from the developers’ master key.

In the open-source world this tends to be done via GnuPG.

Chain of trust

A package manager is a piece of software - if you download and run it then presumably you trust it.

Package developers have to ask the package manager developer to sign their keys.