Security Considerations and Building Trust

Olivier Mehani

Free Software Sydney Meeting — 2015-09-10

About me

- shtrom
- Been using Free software for 15 years (Linux, OpenBSD, . . .)
- Researcher at NICTA in the (former) Network Research Group
- Write code daily
- Administrate various networks at home and at work
- If I can't patch it, I won't use it
- mailto:shtrom@ssji.net
- 4435 CF6A 7C8D DD9B E2DE F5F9 F012 A6E2 98C6 6655
- http://blog.narf.ssji.net;
  http://www.narf.ssji.net/~shtrom/wiki/

About me

Outline

Hashing

Assymetric Cryptography
- Pretty Good Privacy (PGP)/Gnu Privacy Guard (GPG)
- Public-Key Infrastructure (PKI)

Trusting trust

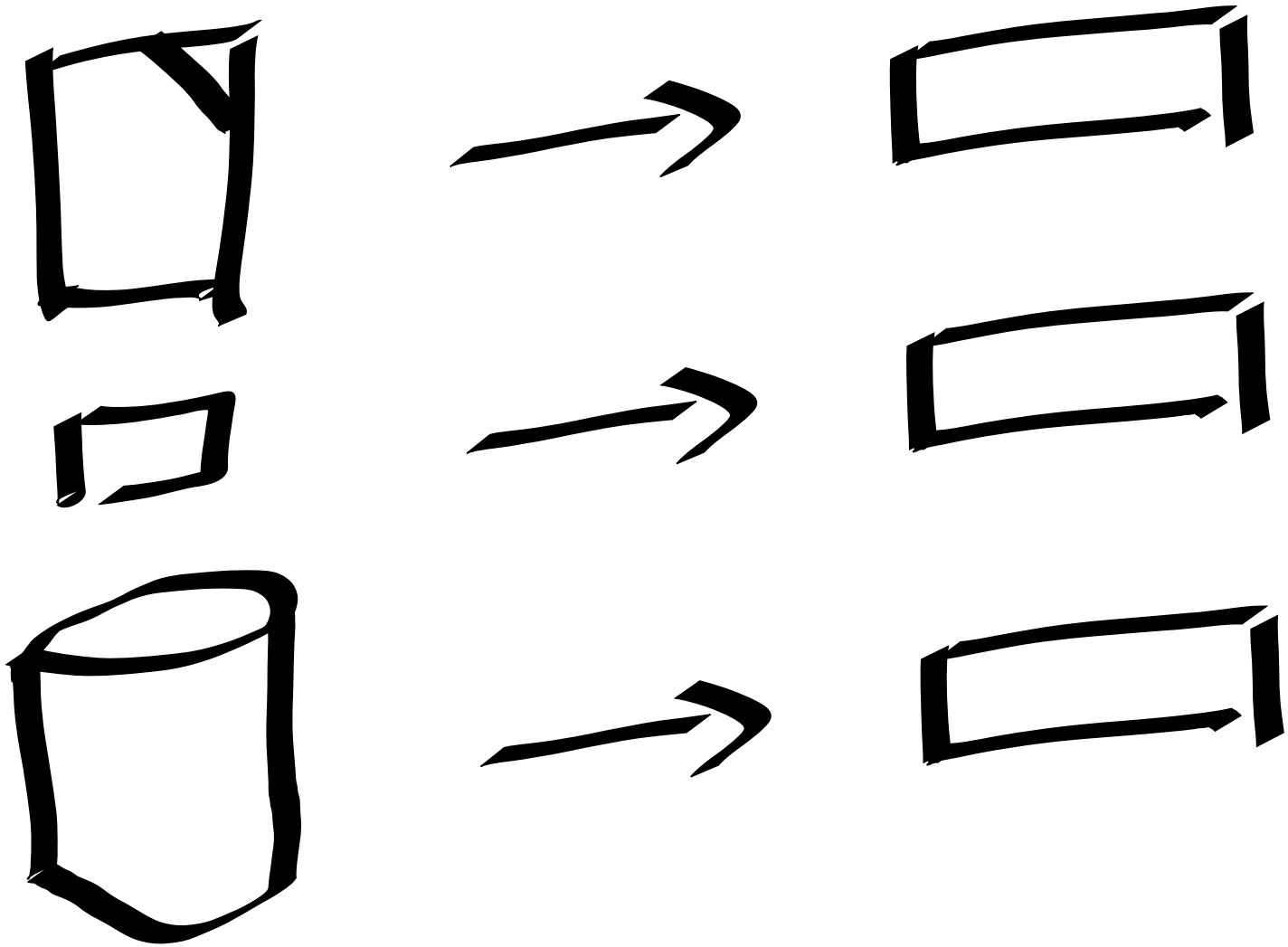
Reproducible builds

Conclusion

Hashing

Basic multitool

- Summarise arbitrary length of data into a small fixed size



Hashing

Basic multitool

- Summarise arbitrary length of data into a small fixed size
- Many applications
  - Efficient data structures: search for hash rather than full contents
    - Hash tables

Hashing

Basic multitool

- Summarise arbitrary length of data into a small fixed size
- Many applications
  - Efficient data structures: search for hash rather than full contents
    - Hash tables
  - Content adressing: search for file content locally or remotely
    - Git, BitTorrent

Hashing

Basic multitool

- Summarise arbitrary length of data into a small fixed size
- Many applications
  - Efficient data structures: search for hash rather than full contents
    - Hash tables
  - Content adressing: search for file content locally or remotely
    - Git, BitTorrent
  - Verify *integrity*: hash matches downloaded file
    - `md5sum`, `sha1sum`, `sha256sum`

Hashing

Basic multitool

- Summarise arbitrary length of data into a small fixed size
- Many applications
  - Efficient data structures: search for hash rather than full contents
    - Hash tables
  - Content adressing: search for file content locally or remotely
    - Git, BitTorrent
  - Verify *integrity*: hash matches downloaded file
    - `md5sum`, `sha1sum`, `sha256sum`
- Cryptographic hash

Hashing

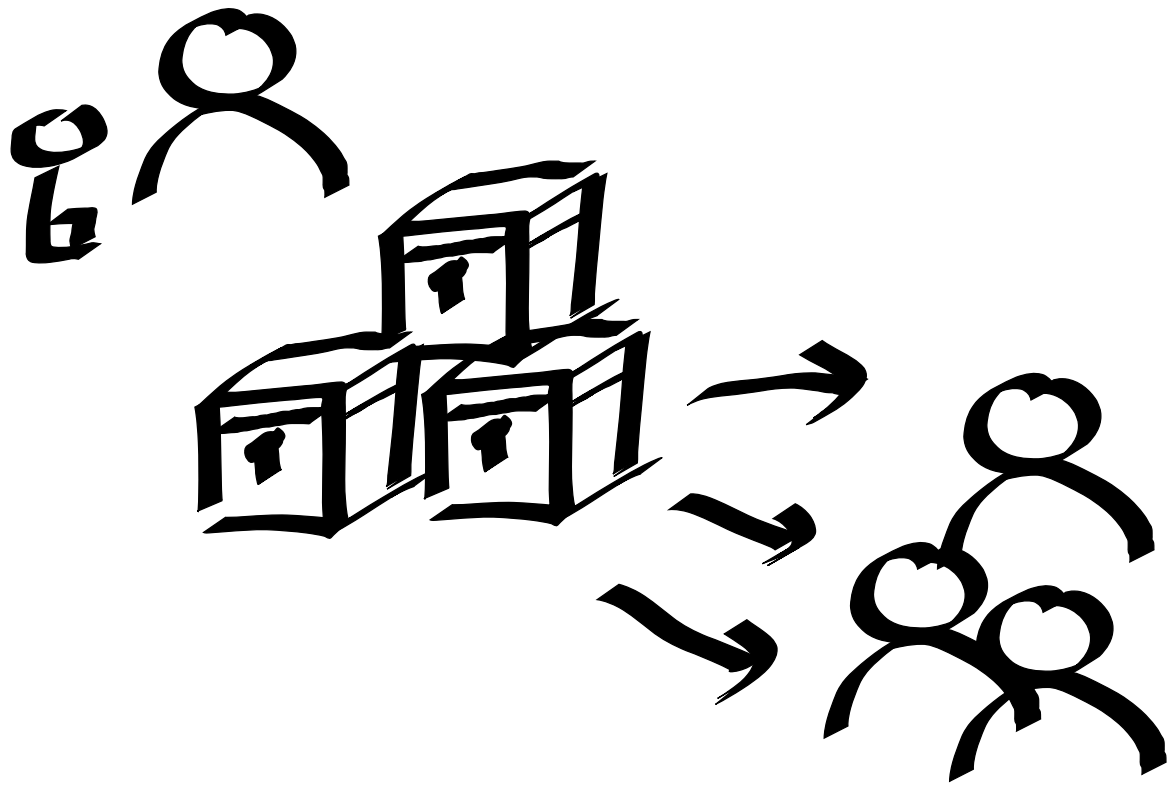
Basic multitool

- Summarise arbitrary length of data into a small fixed size
- Many applications
  - Efficient data structures: search for hash rather than full contents
    - Hash tables
  - Content adressing: search for file content locally or remotely
    - Git, BitTorrent
  - Verify *integrity*: hash matches downloaded file
    - `md5sum`, `sha1sum`, `sha256sum`
- Cryptographic hash
  - Easy to verify that data matches
  - *Hard* to create data matching a specific hash
  - $\Rightarrow$ Block chains proof-of-work
    - Brute-force a random value for a block which makes the hash start with n 0s

Assymetric Cryptography

Public/private keypairs

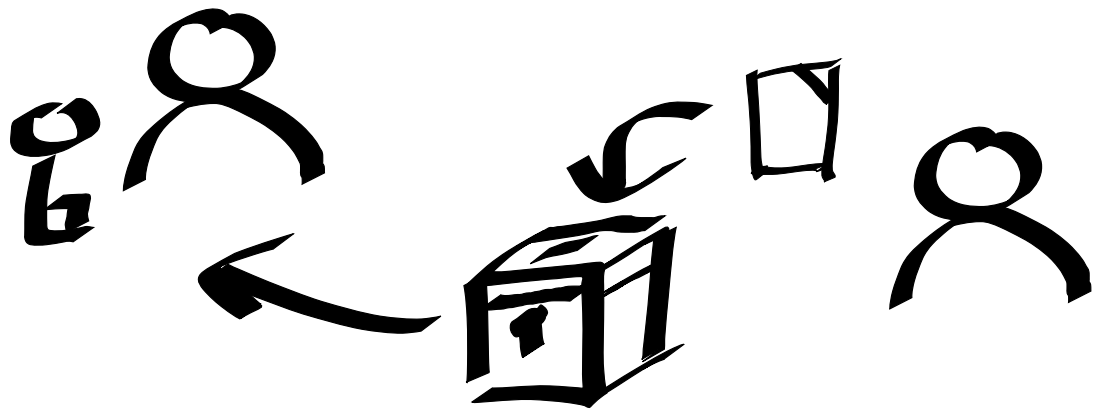
- Key pair
  - Publish the public key widely
  - Keep the private key safe



Assymetric Cryptography

Public/private keypairs

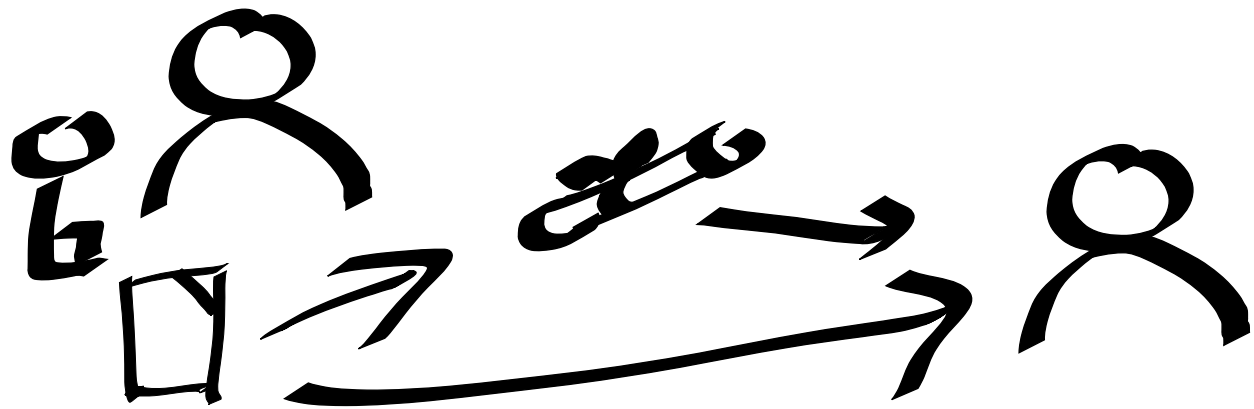
- Key pair
  - Publish the public key widely
  - Keep the private key safe
- Two primitives
  - Encryption to a recipient
    - Use the recipient's public key to generate ciphertext
    - Use the recipient's private key to decrypt ciphertext



Assymetric Cryptography

Public/private keypairs

- Key pair
  - Publish the public key widely
  - Keep the private key safe
- Two primitives
  - Encryption to a recipient
    - Use the recipient's public key to generate ciphertext
    - Use the recipient's private key to decrypt ciphertext
  - Signature from a sender
    - Use the sender's private key to encrypt a *hash* of the content
    - Use the sender's public key to decrypt the hash, and verify that it matches the content



Assymetric Cryptography

Public/private keypairs

- Key pair
  - Publish the public key widely
  - Keep the private key safe
- Two primitives
  - Encryption to a recipient
    - Use the recipient's public key to generate ciphertext
    - Use the recipient's private key to decrypt ciphertext
  - Signature from a sender
    - Use the sender's private key to encrypt a *hash* of the content
    - Use the sender's public key to decrypt the hash, and verify that it matches the content
    - $\Rightarrow$ Can now check *authenticity* of data

Assymetric Cryptography

Public/private keypairs

- Key pair
  - Publish the public key widely
  - Keep the private key safe
- Two primitives
  - Encryption to a recipient
    - Use the recipient's public key to generate ciphertext
    - Use the recipient's private key to decrypt ciphertext
  - Signature from a sender
    - Use the sender's private key to encrypt a *hash* of the content
    - Use the sender's public key to decrypt the hash, and verify that it matches the content
    - $\Rightarrow$ Can now check *authenticity* of data

- Problem: How do we know who a public key really belongs to?

Pretty Good Privacy (PGP)/Gnu Privacy Guard (GPG)

Building a decentralised Web-of-trust

- OpenPGP certificates binding some identity to a key pair

Pretty Good Privacy (PGP)/Gnu Privacy Guard (GPG)

Building a decentralised Web-of-trust

- OpenPGP certificates binding some identity to a key pair
  - Described by their fingerprint
    - "Kinda like a hash"
    - `gpg --fingerprint f012a6e298c66655`

Pretty Good Privacy (PGP)/Gnu Privacy Guard (GPG)

Building a decentralised Web-of-trust

- OpenPGP certificates binding some identity to a key pair
  - Described by their fingerprint
    - "Kinda like a hash"
    - `gpg --fingerprint f012a6e298c66655`
- Verify a PGP certificate
  - First-hand: Verify that the fingerprint match what the owner says
    - *over a trustworthy channel* (*e.g.*, signing party)
    - then sign and publish the signature for others to check

Pretty Good Privacy (PGP)/Gnu Privacy Guard (GPG)

Building a decentralised Web-of-trust

- OpenPGP certificates binding some identity to a key pair
  - Described by their fingerprint
    - "Kinda like a hash"
    - `gpg --fingerprint f012a6e298c66655`
- Verify a PGP certificate
  - First-hand: Verify that the fingerprint match what the owner says
    - *over a trustworthy channel* (*e.g.*, signing party)
    - then sign and publish the signature for others to check
  - Second-hand: Check that enough turstworthy users have signed the certificate

Pretty Good Privacy (PGP)/Gnu Privacy Guard (GPG)

Building a decentralised Web-of-trust

- OpenPGP certificates binding some identity to a key pair
  - Described by their fingerprint
    - "Kinda like a hash"
    - `gpg --fingerprint f012a6e298c66655`
- Verify a PGP certificate
  - First-hand: Verify that the fingerprint match what the owner says
    - *over a trustworthy channel* (*e.g.*, signing party)
    - then sign and publish the signature for others to check
  - Second-hand: Check that enough turstworthy users have signed the certificate
  - Trust on first sight, trust most used
    - Not really sure...

Pretty Good Privacy (PGP)/Gnu Privacy Guard (GPG)

Building a decentralised Web-of-trust

- OpenPGP certificates binding some identity to a key pair
  - Described by their fingerprint
    - "Kinda like a hash"
    - `gpg --fingerprint f012a6e298c66655`
- Verify a PGP certificate
  - First-hand: Verify that the fingerprint match what the owner says
    - *over a trustworthy channel* (*e.g.*, signing party)
    - then sign and publish the signature for others to check
  - Second-hand: Check that enough turstworthy users have signed the certificate
  - Trust on first sight, trust most used
    - Not really sure...
- Sign and encrypt any data/message (email, jabber, ...)

Pretty Good Privacy (PGP)/Gnu Privacy Guard (GPG)

Building a decentralised Web-of-trust

- OpenPGP certificates binding some identity to a key pair
  - Described by their fingerprint
    - "Kinda like a hash"
    - `gpg --fingerprint f012a6e298c66655`
- Verify a PGP certificate
  - First-hand: Verify that the fingerprint match what the owner says
    - *over a trustworthy channel* (*e.g.*, signing party)
    - then sign and publish the signature for others to check
  - Second-hand: Check that enough turstworthy users have signed the certificate
  - Trust on first sight, trust most used
    - Not really sure...
- Sign and encrypt any data/message (email, jabber, ...)
- Check data authenticity
  - Verify signature of a hash that matches dowloaded data
    - `gpg --verify data.asc`

Public-Key Infrastructure (PKI)

- Trusted third parties: Certificate authorities (CA)

[1] http://cacert.org
[2] http://letsencrypt.org/

Public-Key Infrastructure (PKI)

- Trusted third parties: Certificate authorities (CA)
  - Root certificate trusted by clients (*e.g.*, browsers' trust stores)
  - Sign other certificates after verifying who they belong to (*e.g.*, domain owner)

[1] http://cacert.org

[2] http://letsencrypt.org/

Public-Key Infrastructure (PKI)

- Trusted third parties: Certificate authorities (CA)
  - Root certificate trusted by clients (*e.g.*, browsers' trust stores)
  - Sign other certificates after verifying who they belong to (*e.g.*, domain owner)
    - and get money along the way

[1] http://cacert.org

[2] http://letsencrypt.org/

Public-Key Infrastructure (PKI)

- Trusted third parties: Certificate authorities (CA)
  - Root certificate trusted by clients (*e.g.*, browsers' trust stores)
  - Sign other certificates after verifying who they belong to (*e.g.*, domain owner)
    - and get money along the way
  - Problem: *any* dodgy CA in the trust store can issue a validable certificate for *any* domain
    - $\Rightarrow$ broken model
    - DNSSEC/DANE *might* help reduce the attack surface

[1] http://cacert.org

[2] http://letsencrypt.org/

Public-Key Infrastructure (PKI)

- Trusted third parties: Certificate authorities (CA)
  - Root certificate trusted by clients (*e.g.*, browsers' trust stores)
  - Sign other certificates after verifying who they belong to (*e.g.*, domain owner)
    - and get money along the way
  - Problem: *any* dodgy CA in the trust store can issue a validable certificate for *any* domain
    - $\Rightarrow$ broken model
    - DNSSEC/DANE *might* help reduce the attack surface
- Alternate CAs models
  - CAcert:[1] based on web-of-trust verification
    - human assurers verify your name/ID
    - not in common truststores
  - Let's Encrypt:[2] Mozilla and others' initiative
    - reduce the barrier to entry for encryption
    - doesn't solve the trust abuse problem
    - will launch soon

[1] http://cacert.org

[2] http://letsencrypt.org/

Trusting trust

- One bit flip can introduce a vulnerability
  - Hashes can help identify this

[3] K. Thompson. "Reflections on Trusting Trust". In: *Communications of the ACM* 27.8 (Aug. 1984). Ed. by P. J. Denning, pp. 761–763. ISSN: 0001-0782. DOI: 10.1145/358198.358210. URL: http://www.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf

Trusting trust

- One bit flip can introduce a vulnerability
  - Hashes can help identify this
- Compiler/toolchain can be compromised[3]
  - Source code is clean
  - Binary isn't

[3] K. Thompson. "Reflections on Trusting Trust". In: *Communications of the ACM* 27.8 (Aug. 1984). Ed. by P. J. Denning, pp. 761–763. ISSN: 0001-0782. DOI: 10.1145/358198.358210. URL: http://www.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf

Trusting trust

- One bit flip can introduce a vulnerability
  - Hashes can help identify this
- Compiler/toolchain can be compromised[3]
  - Source code is clean
  - Binary isn't
- $\Rightarrow$ Seeing the source and trusting the build system is not enough

[3] K. Thompson. "Reflections on Trusting Trust". In: *Communications of the ACM* 27.8 (Aug. 1984). Ed. by P. J. Denning, pp. 761–763. ISSN: 0001-0782. DOI: 10.1145/358198.358210. URL: http://www.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf

Reproducible builds

- Don't trust a single party

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary
  - . . . but what if binary not built from published version of the code?

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary
  - . . . but what if binary not built from published version of the code?
- Trust packager (*e.g.*, Debian, F-Droid)
  - Take source code, build binaries
  - Hash & sign binary

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary
  - . . . but what if binary not built from published version of the code?
- Trust packager (*e.g.*, Debian, F-Droid)
  - Take source code, build binaries
  - Hash & sign binary
  - . . . but what if key gets stolen?

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary
  - . . . but what if binary not built from published version of the code?
- Trust packager (*e.g.*, Debian, F-Droid)
  - Take source code, build binaries
  - Hash & sign binary
  - . . . but what if key gets stolen?
  - . . . or what if the build machine is compromised?

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary
  - . . . but what if binary not built from published version of the code?
- Trust packager (*e.g.*, Debian, F-Droid)
  - Take source code, build binaries
  - Hash & sign binary
  - . . . but what if key gets stolen?
  - . . . or what if the build machine is compromised?
  - . . . or what if the packager is not trustworthy?

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary
  - . . . but what if binary not built from published version of the code?
- Trust packager (*e.g.*, Debian, F-Droid)
  - Take source code, build binaries
  - Hash & sign binary
  - . . . but what if key gets stolen?
  - . . . or what if the build machine is compromised?
  - . . . or what if the packager is not trustworthy?
- Reproducible builds!

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary
  - . . . but what if binary not built from published version of the code?
- Trust packager (*e.g.*, Debian, F-Droid)
  - Take source code, build binaries
  - Hash & sign binary
  - . . . but what if key gets stolen?
  - . . . or what if the build machine is compromised?
  - . . . or what if the packager is not trustworthy?
- Reproducible builds!
  1. Developer publishes source code
  2. . . . builds binary, and create and publish signature

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary
  - . . . but what if binary not built from published version of the code?
- Trust packager (*e.g.*, Debian, F-Droid)
  - Take source code, build binaries
  - Hash & sign binary
  - . . . but what if key gets stolen?
  - . . . or what if the build machine is compromised?
  - . . . or what if the packager is not trustworthy?
- Reproducible builds!
  1. Developer publishes source code
  2. . . . builds binary, and create and publish signature
  3. Packager takes source code
  4. . . . rebuilds binary, and check that developer's signature matches

Reproducible builds

- Don't trust a single party
- Trust uncompiled source code
  - Hopefully seen by many eyeballs
- Trust developer
  - Hash & sign binary
  - . . . but what if binary not built from published version of the code?
- Trust packager (*e.g.*, Debian, F-Droid)
  - Take source code, build binaries
  - Hash & sign binary
  - . . . but what if key gets stolen?
  - . . . or what if the build machine is compromised?
  - . . . or what if the packager is not trustworthy?
- Reproducible builds!
  1. Developer publishes source code
  2. . . . builds binary, and create and publish signature
  3. Packager takes source code
  4. . . . rebuilds binary, and check that developer's signature matches
  5. Anybody else can redo it and verify independently

Conclusion

Tools to build trust

- Hashes: Data summary and integrity

Conclusion

Tools to build trust

- Hashes: Data summary and integrity
- Cryptography: Verify authenticity
  - Decentralised trust: PFP WoT
  - Centralised, and brittle, approach: SSL PKI

Conclusion

Tools to build trust

- Hashes: Data summary and integrity
- Cryptography: Verify authenticity
  - Decentralised trust: PFP WoT
  - Centralised, and brittle, approach: SSL PKI
- Reproducible builds
  - No single source of trust, every step verifiable

Conclusion

Tools to build trust

- Hashes: Data summary and integrity
- Cryptography: Verify authenticity
  - Decentralised trust: PFP WoT
  - Centralised, and brittle, approach: SSL PKI
- Reproducible builds
  - No single source of trust, every step verifiable
- Get in the habit daily
  - GPG for email, OTR for chat
  - Check hashes and signatures when downloading files

Conclusion

Tools to build trust

- Hashes: Data summary and integrity
- Cryptography: Verify authenticity
  - Decentralised trust: PFP WoT
  - Centralised, and brittle, approach: SSL PKI
- Reproducible builds
  - No single source of trust, every step verifiable
- Get in the habit daily
  - GPG for email, OTR for chat
  - Check hashes and signatures when downloading files

Conclusion

Tools to build trust

- Hashes: Data summary and integrity
- Cryptography: Verify authenticity
  - Decentralised trust: PFP WoT
  - Centralised, and brittle, approach: SSL PKI
- Reproducible builds
  - No single source of trust, every step verifiable
- Get in the habit daily
  - GPG for email, OTR for chat
  - Check hashes and signatures when downloading files
  - Verify fingerprints and sign keys

Conclusion

Tools to build trust

- Hashes: Data summary and integrity
- Cryptography: Verify authenticity
  - Decentralised trust: PFP WoT
  - Centralised, and brittle, approach: SSL PKI
- Reproducible builds
  - No single source of trust, every step verifiable
- Get in the habit daily
  - GPG for email, OTR for chat
  - Check hashes and signatures when downloading files
  - Verify fingerprints and sign keys
  - Try to rebuild reproducible packages from the Debian archive!