ZTRUST:
Anchoring the IBM Z Supply Chain

Rick Troth, VSSI
<richard.troth@vsoftsys.com>

Brian Hugenbruch, IBM
<bwhugen@us.ibm.com>

the gang

- Matt Hogstrom (z/OS guy)
- Rick Troth (z/VM guy)
- Brian Hugenbruch (inside man at IBM)
- Berry van Sleeuwen (Dutch delegation)
- Mike MacIsaac (maint master)
- Lyz Joseph (Linux lass)

IBM Z community members

(actual humans)

Agenda

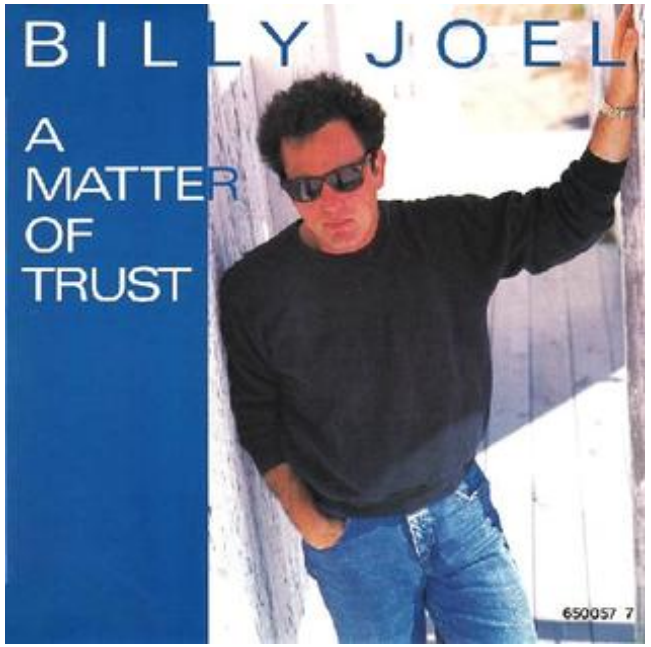
- The need: supply chain, provenance, attestation
- Introducing ZTRUST
  - The PKI method
  - The PGP method
- Why it matters
- Where ZTRUST is today
- Where we’re headed

Securing the Supply Chain

- Industry demand for Code Signing
  - Driven by regulatory compliance
    (EU CRA, White House Executive Orders, PCI DSS, more)
  - Driven by public embarrassment (e.g. SolarWinds)

- Answers the question, “is this code from a trusted source?”

It’s All About Trust

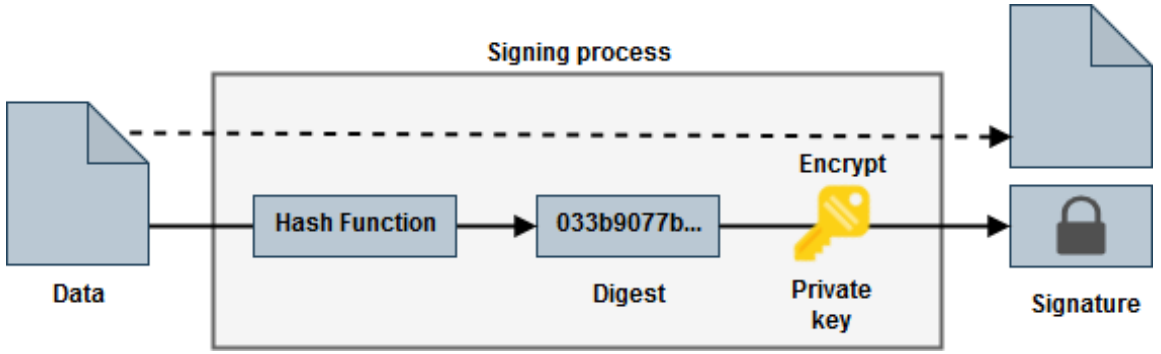


Keys are cheap.

It’s trust that matters.

What's in a Signature? (Part 1)

- A digital signature is an encrypted hash or checksum
  - Like HTTPS for websites, except for at-rest code
  - The hash provides integrity (no one changed a single bit)
  - The encryption provides identity and non-repudiation



The why and how of signatures (1)

Encryption is based on mathematical secrets

- **Symmetric**: same key locks and unlocks
- **Asymmetric**: key *pair* -- one locks, other unlocks
  - Keep the locking key private
  - Share the public key widely

If we can decrypt a binary with Rick's public key,
we know it came from Rick – only Rick has Rick's *private* key

- If he's protected it properly, of course

Same logic and math for both PKI and PGP

The why and how of signatures (2)

The builder:

- Hashes the payload
- Encrypt hash with private key we have a signature!
- Sends the payload and the signature out into the world

often done in one fell swoop

The user:

- Decrypts signature with public key we have the hash!
- Compares the decrypted hash to a new, local hash of payload
-

Same logic, same math, for both PKI and PGP

How digital signatures help

Hashing (a cryptographic checksum) provides integrity validation

One way function with no collisions

If you hash data twice, the result is always the same

If the data is modified by even one bit, result is (often wildly) different

Public-private key encryption provides authenticity

If you encrypt a string with a private key, anyone with the public key knows for certain it came from you

[Not a factor here, but] If you encrypt code with a public key, only the person with the private key can read it.

What signatures don't do

Prevent	Prevent access to images •Your local access policies prevent this (this is RACF and similar)
Prevent	Prevent tampering after validation •Your local access policies prevent this •Digital signature verification will detect tampered code on the next verify, though
Prevent	Prevent authorized changes •You do have to re-sign content after you update! •Because you need a private key to do this, best to do this on a very secure system

Key and Certificate Lifetimes

Expirations versus Revocations

- Certificates can expire (like a carton of milk)
- Certificates can be revoked (like throwing Alan Altmark out of a party) (he knows what he did)

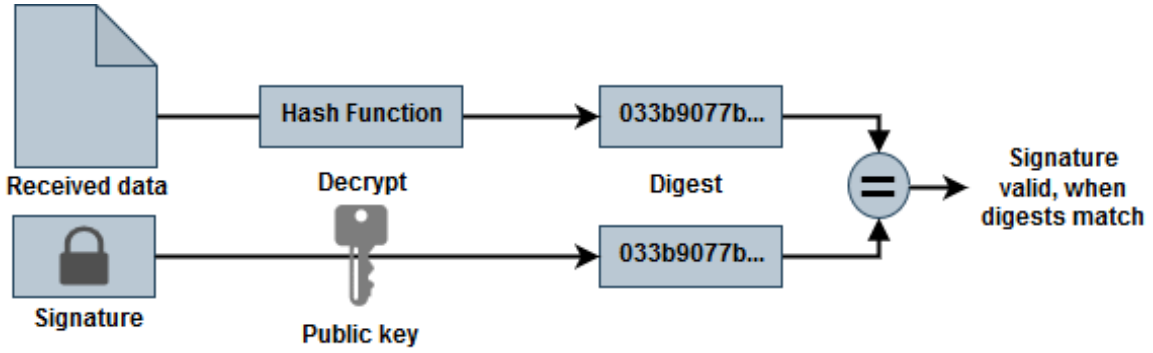
Is there a “better way”? **Expiration dates** versus **revocation lists**

- CRLs and OCSP often require an internet connection (go check the guest list)
- Not every technical implementation will check if the cert has expired (eww)

What's in a Signature? (Part 2)

- The goal (revisited): meet industry demand for **tamper-resistant code** from a **reputable source**

- On the receiving end …

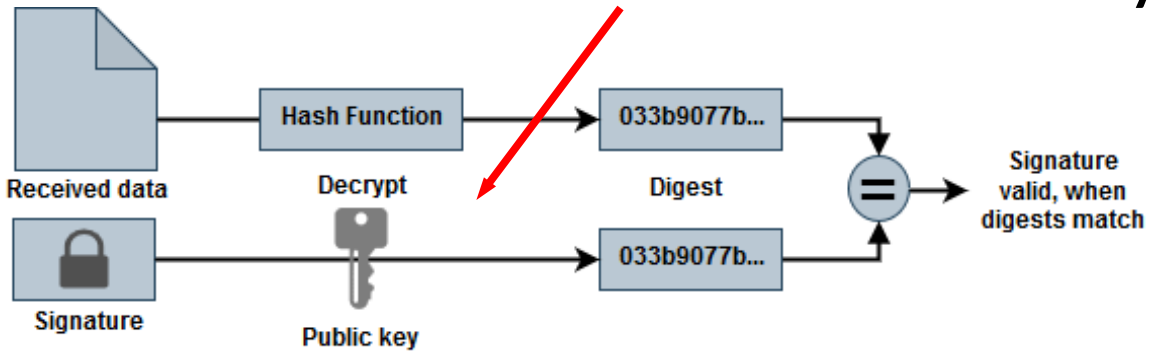


What's in a Signature? (Part 2)

- The goal (revisited): meet industry demand
  for **tamper-resistant code** from a **reputable source**

- On the receiving end …

where'd they get that key?



Problem Statement: Who Signs What?

In the case of vendors, this is easy

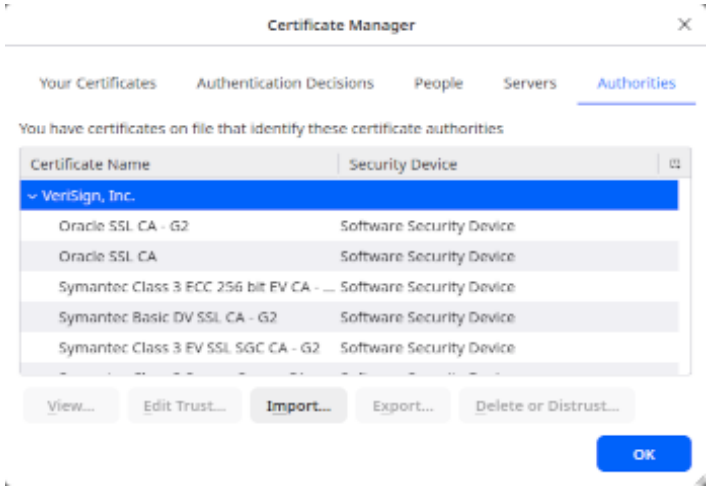
- VSSI signs VSSI code
- IBM signs IBM code
- Red Hat signs Red Hat code
- …and so on

But what about:

- Your CP Exits
  or local modifications
- Actively maintained utilities
  on the VM Download page
  or out in Github
- More storied hacks suffering
  from author existence failure

ZTRUST

- A “trust anchor” for the IBM Z community
  - PKI root certificates
  - And also also PGP keys
- Uploaded to GitHub for reachability
- Part of the Open Mainframe Project (OMP)
  - Gives it an official home
  - A focal point for **starting** the trust discussion

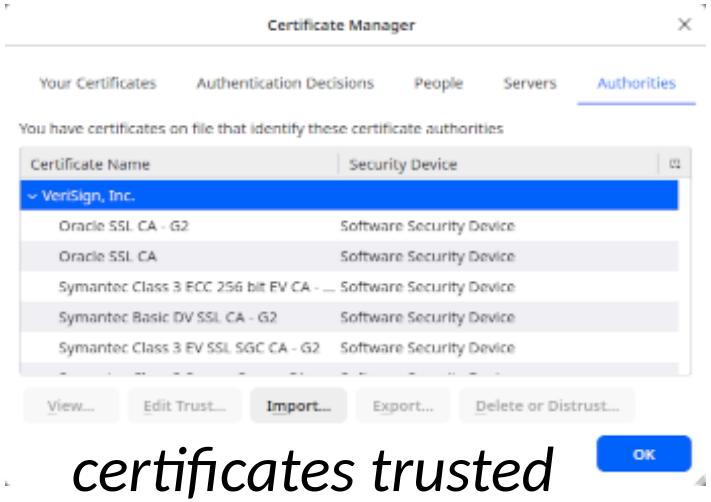


The PKI Trust Model

- The “Cathedral” approach: *trust by recognized authority*
-
- Specific Certificate Authorities recognized … thus trusted
  - business trust / procedural trust
  - “being your own CA” sometimes fine for test … less so for prod
- Root certificates (from CAs) pre-loaded into (e.g.) browser
  - cryptographic trust / operational trust
- Some verification (e.g. web browsing) is automatic

The PKI Method

- Producer **signs** deliverables
  - A specific “code signing” digital certificate
  - Private key kept under wraps
  - Public-facing certificate available for use
- Consumers **verifies** using these certs
  - Ideally, these are preloaded
  - Some manual loading may be required (IBM)
  - Verification only works
    if that certificate is available on your system



certificates trusted by their presence

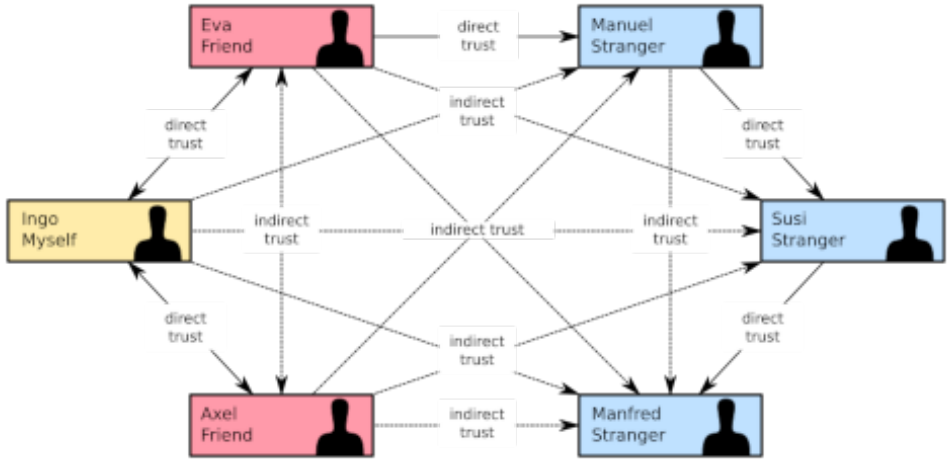
The PKI Method in Action: z17

- **Given**: Appropriate hardware IBM z16 with appropriate service or an IBM z17
- **Given**: IBM or Linux certificate(s) loaded into HMC and associated with LPARs
- **Given**: list-directed IPL (check the “Secure” box, issue SET LOADDEV…)
- **Then**:
  - Signed modules are validated at time of boot
    - **Today**: z/OS, Linux kernel and packages running in LPAR or as z/VM guests
    - **Tomorrow**: z/VM V7.5 SAPL and CPLOAD
  - Boot fails with wait state if security is required and either
    (a) appropriate certificate is not found or (b) hash comparison fails

The PGP Method

- The web-of-trust method: *trust by community consensus*
-
- Producer creates a “key pair”
- Producer “signs” deliverables
- Consumers obtain the public key
- Optionally collect other PGP keys and vet the producer’s key

keys are cross-signed
trusted by signatures



The PGP Trust Model

- Follow the chain of trust
- Possible multiple “paths”

from	**stats** Rick Troth <rmt.at.casita.net>	96af6544edf138d9
to	**stats** Nick Mathewson <nickm.at.alum.mit.edu>	fe43009c4607b1fb
find	**reverse path**	trust paths
see also	The data on this page is available as a **json file**.	reset

0 96af6544edf138d9 **stats** Rick Troth <rmt.at.casita.net> #10982 *signs*
1 8a3171ef366150ce **stats** David Steele <steele.at.debian.org> #4667 *signs*
2 8cbf9a322861a790 **stats** Micah Anderson <micah.at.riseup.net> #218 *signs*
3 fe43009c4607b1fb **stats** Nick Mathewson <nickm.at.alum.mit.edu> #5684

0 96af6544edf138d9 **stats** Rick Troth <rmt.at.casita.net> #10982 *signs*
1 9ec002fe1c9ca517 **stats** Michael C. Schultheiss <schultmc.at.debian.org> #460 *signs*
2 06eaa066e397832f **stats** Luca Capello <luca.at.pca.it> #21 *signs*
3 65b3f094ea3e4d61 **stats** Jens Kubieziel <jens.at.kubieziel.de> #274 *signs*
4 fe43009c4607b1fb **stats** Nick Mathewson <nickm.at.alum.mit.edu> #5684

0 96af6544edf138d9 **stats** Rick Troth <rmt.at.casita.net> #10982 *signs*
1 600a553ff666c91d **stats** Jeff Licquia <jeff.at.licquia.org> #889 *signs*
2 89cd4b21607559e6 **stats** Benjamin Hill (Mako) <mako.at.atdot.cc> #7 *signs*
3 42e86a2a11f48d36 **stats** David Goulet <dgoulet.at.ev0ke.net> #775 *signs*
4 fe43009c4607b1fb **stats** Nick Mathewson <nickm.at.alum.mit.edu> #5684

How Do You Know?

- PKI root certificate in the “store” – trusted
- PGP public key on the “keyring” – trusted
- PKI root certificate obtained from the web
- PKI root certificates pre-loaded
- PGP key retrieved from a key server
- PGP key signed by someone you know

in-person (dancing optional)

…

… but what if you can’t?

Combined Capabilities

- PGP keys cross-signed for “web of trust” attestation
- PGP keys cross-signed by Z Community members
- PGP keys in ZTRUST (trusted as a bundle)
  - in addition to keys you may already have
- PKI root certs signed with PGP keys
- PKI root certs in ZTRUST (trusted as a bundle)
  - in addition to root certificates you may already have

Where we are with ZTRUST

- Embryonic Trust Anchor
- Introducing the Project
- Collecting Keys and Certificates
- Establishing Community Connections
-

“Cryptography is Easy; Key Management is Hard”

What's Next for ZTRUST

- Call to action – Publicize the Project
- Document the Details (how-to and etc)
- Collect More Keys and Certificates
- Adopt and/or Develop Scripting
- Push for Signing of Volunteer-Provided Projects

Rick Troth, VSSI
<richard.troth@vsoftsys.com>

Brian Hugenbruch, IBM
<bwhugen@us.ibm.com>

Getting involved:

`https://github.com/openmainframeproject/wg-ztrust/`

END