cpackget Signed Packs feature preview

Luís Tonicha
25/10/2022

Linaro

Overview

- Problem
- Solution
- Implementation
- Attack vector analysis
- Pros and Cons
- Future work and Requirements

Problem

No integrity check

Packs are installed “as-is”, with no defined way of checking if the download was successful, or if the pack is corrupted.

No authenticity guarantees

Installing a pack, whether from a public index or another location, requires blindly trusting that it came from the right origin (the vendor).

Lacking Access Control

Other than manually accepting/rejecting packs, it’s not possible to setup a vendor “allow/denylist” based on a trusted entity.

Solution

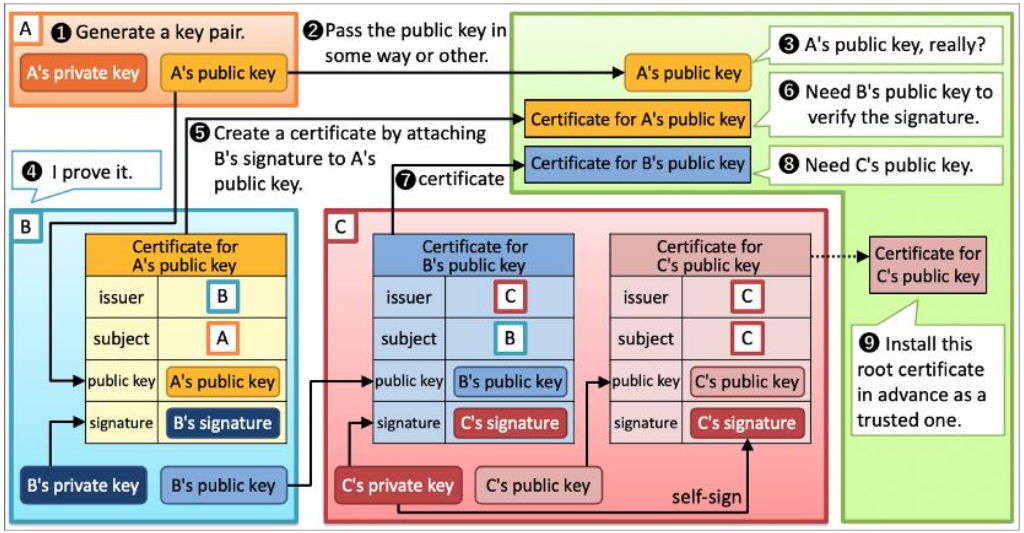
Public Key Infrastructure / X.509

X.509 public key certificates are the industry standard to provide cryptographic security guarantee by establishing a centralized chain of trust between participating entities.

PGP (Pretty Good Privacy) signatures

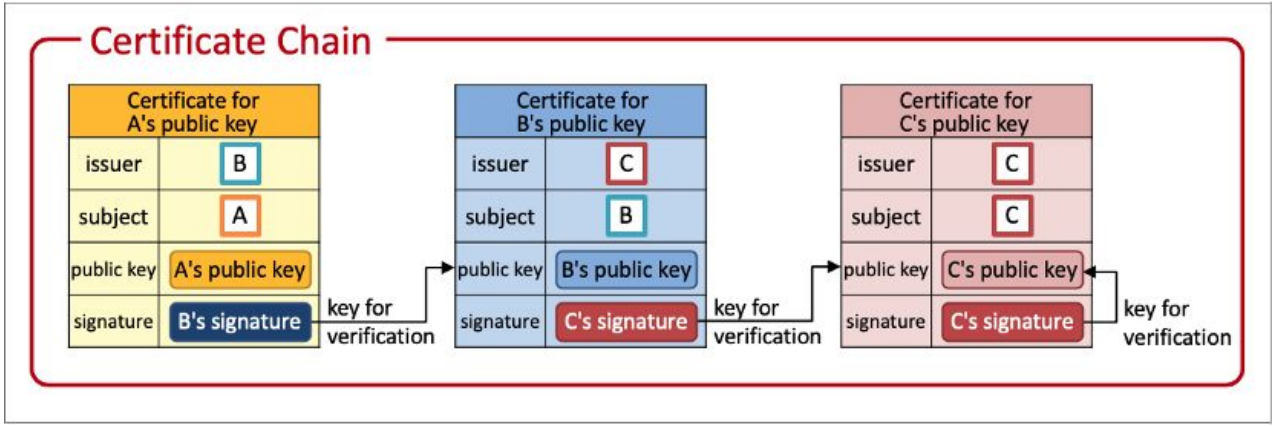
Using a web of trust model, PGP can also provide the same guarantees, as long as the end user trusts the public key distribution mechanism.

Solution - X.509



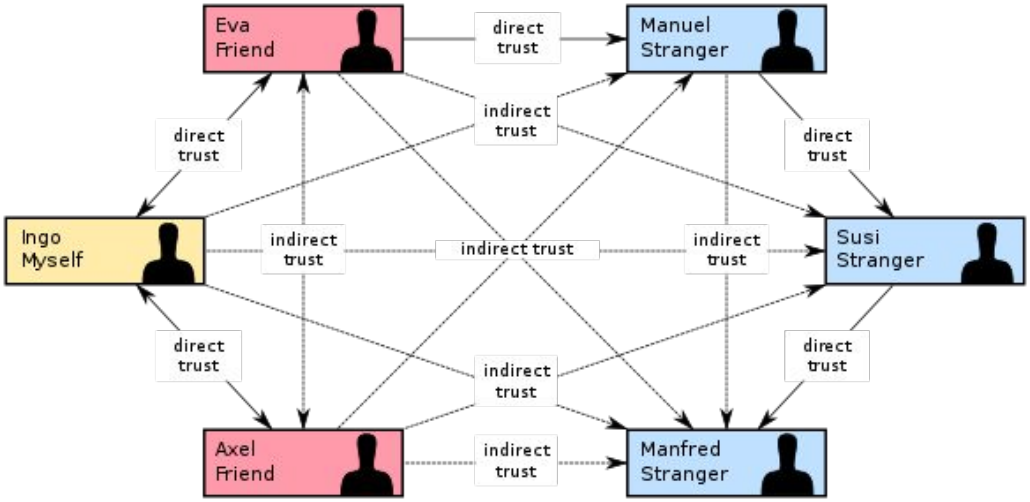
X.509 issuance steps

Solution - X.509



X.509 certificate chain (leaf, intermediate and root certificates)

Solution - PGP



PGP web-of-trust example

Implementation - Concepts

Use the underlying .zip format

Open-CMSIS-Pack's *.pack* extension is a "disguised" Zip file, which always has a general comment field, typically left untouched.

Hash the contents of the pack

Using the industry standard SHA256 hashing algorithm, cpackget hashes the contents (the actual files, not the final .pack), producing one final, constant size hash.

Signature scheme/tag

We define a fixed signature scheme (which represents these objects) that is easily verifiable and lightweight to compute.

Implementation - 3 different modes

“Full” (default one)

Requires a X.509 public key certificate and its private key to sign. The former is shipped alongside the signed hash, used to verify each other and the pack’s contents. No additional user steps are needed to verify the pack (other than the pack itself).

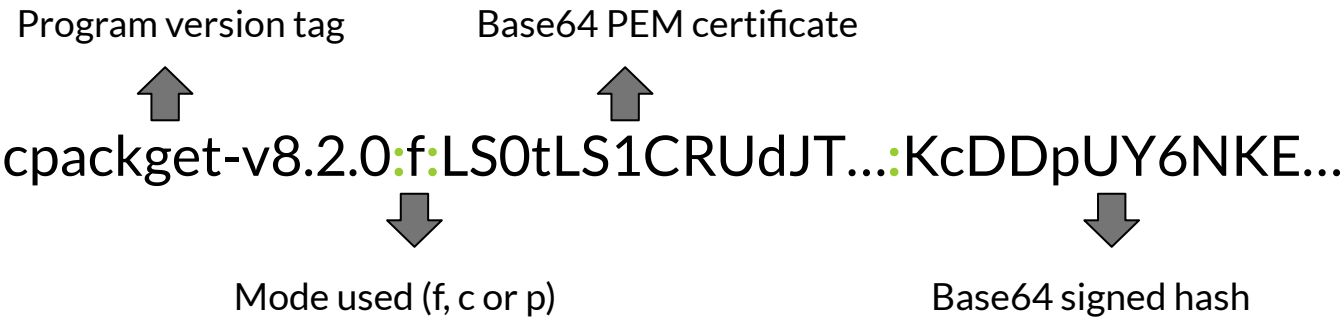
“PGP”

Using an armored (passphrase protected) PGP private key, the contents are signed and this signature is embed and shipped in the pack. To verify, the user must provide the PGP public key that it’s checking against (this can be easily automated).

“Cert-only”

Only embeds & ships the vendor’s certificate. Does not provide security guarantees if the user does not trust the download/ install channel. Best used to implement a simple client side Access Control List.

Implementation - Signature scheme



(slightly varies for *p* and *c* modes)

Implementation - Example uses

$ cpackget signature-create Vendor.Pack.1.0.0.pack -k myprivate.key -c mycert.pem

$ cpackget signature-create --pgp Vendor.Pack.1.0.0.pack -k myprivate.pgp

$ cpackget signature-create --cert-only Vendor.Pack.1.0.0.pack -c mycert.pem

Implementation - Example uses

$ cpackget signature-verify Vendor.Pack.1.0.0.pack.signed

$ cpackget signature-verify Vendor.Pack.1.0.0.pack.signed -k pubkey.pgp

$ cpackget signature-verify Vendor.Pack.1.0.0.pack.signed –e # (only exports the cert.)

(upon feature deployment, these would be automatically called when installing a pack)

Attack vector analysis

```
Archive:  Vendor.Pack.1.0.0.pack.signed
v0.8.1-alpha.1-18-
gf0226dc:f:LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUZpekNDQTNPZ0F3SUJBZ0lVQTBXcXhqSE5qZ2plUy9jVn
GDyHKqQN4mhuMq2+Keno38MN/z90x2R7kL7VXo/eqaKSj7MT4vqmbRTkITHXzQM7itnS/M//E4euQSsK3+ULcDi5ag5/
Labp9yV8MhxWYeklfDX8CiD5jplrWvv9BPjbkA48tC5WD6IKkmOkPvGnyqcXvh9p41VzUb8apWPYlWd2eBnuhHDqm0+NHcDr5
Nqv/
rxGmUWxDlrP2zOohdMMJUD6aYpL0GCyXT1SigUrBUHKHStxILAeC4stLldcdScgw1Q7FHlTUqLW8t5Z7WsuZI3E8CTVua1+b0
wNfJRfyJwqzpZJgKmPky/sxr24=
Zip file size: 5774 bytes, number of entries: 12
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-17 17:55 dir/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file1
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file2
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir1/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file3
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file4
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir2/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir2/.gitignore
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir3/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir3/.gitignore
-rw-rw-r--  2.0 unx        0 bX defN 21-Jul-02 02:07 sample_file
-rw-rw-r--  2.0 unx      464 bX defN 21-Jul-13 02:49 Vendor.Pack.pdsc
12 files, 464 bytes uncompressed, 286 bytes compressed:  38.4%
```

Scenario: attacker modifies contents

If it does not modify the signed hash, even changing one bit would get detected as the SHA256 digest wouldn't match.

Result: malicious pack not installed

Attack vector analysis

```
Archive:  Vendor.Pack.1.0.0.pack.signed
v0.8.1-alpha.1-18-
gf0226dc:f:LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUZpekNDQTNPZ0F3SUJBZ0lVQTBXcXhqSE5qZ2plUy9jVn
GDyHKqQN4mhuMq2+Keno38MN/z90x2R7kL7VXo/eqaKSj7MT4vqmbRTkITHXzQM7itnS/M//E4euQSsK3+ULcDi5ag5/
Labp9yV8MhxWYeklfDX8CiD5jplrWvv9BPjbkA48tC5WD6IKkmOkPvGnyqcXvh9p41VzUb8apWPYlWd2eBnuhHDqm0+NHcDr5
Nqv/
rxGmUWxDlrP2zOohdMMJUD6aYpL0GCyXT1SigUrBUHKHStxILAeC4stLldcdScgw1Q7FHlTUqLW8t5Z7WsuZI3E8CTVua1+b0
wNfJRfyJwqzpZJgKmPky/sxr24=
Zip file size: 5774 bytes, number of entries: 12
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-17 17:55 dir/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file1
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file2
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir1/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file3
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file4
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir2/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir2/.gitignore
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir3/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir3/.gitignore
-rw-rw-r--  2.0 unx        0 bX defN 21-Jul-02 02:07 sample_file
-rw-rw-r--  2.0 unx      464 bX defN 21-Jul-13 02:49 Vendor.Pack.pdsc
12 files, 464 bytes uncompressed, 286 bytes compressed:  38.4%
```

Scenario: attacker modifies contents and signed hash

Even if the attacker changes both the contents and signs the hash with its private key, this key will not match the vendor's public key, failing signature verification.

Result: malicious pack not installed

Attack vector analysis

```
Archive:  Vendor.Pack.1.0.0.pack.signed
v0.8.1-alpha.1-18-
gf0226dc:f LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUZpekNDQTNPZ0F3SUJBZ0lVQTBXcXhqSE5qZ2plUy9jVr
GDyHKqQN4mhuMq2+Keno38MN/z90x2R7kL7VXo/eqaKSj7MT4vqmbRTkITHXzQM7itnS/M//E4euQSsK3+ULcDi5ag5/
Labp9yV8MhxWYeklfDX8CiD5jplrWvv9BPjbkA48tC5WD6IKkmOkPvGnyqcXvh9p41VzUb8apWPYlWd2eBnuhHDqm0+NHcDr5
Nqv/
rxGmUWxDlrP2zOohdMMJUD6aYpL0GCyXT1SigUrBUHKHStxILAeC4stLldcdScgw1Q7FHlTUqLW8t5Z7WsuZI3E8CTVua1+b0
wNfJRfyJwqzpZJgKmPky/sxr24=
Zip file size: 5774 bytes, number of entries: 12
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-17 17:55 dir/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file1
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file2
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir1/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file3
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file4
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir2/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir2/.gitignore
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir3/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir3/.gitignore
-rw-rw-r--  2.0 unx        0 bX defN 21-Jul-02 02:07 sample_file
-rw-rw-r--  2.0 unx      464 bX defN 21-Jul-13 02:49 Vendor.Pack.pdsc
12 files, 464 bytes uncompressed, 286 bytes compressed:  38.4%
```

Scenario: attacker modifies contents, signed hash and X.509 certificate, signed by an untrustworthy CA

If all 3 get compromised, an attacker would sign the hash with a phony CA, and ship its certificate with it. cpackget detects that it's not its elected/allowed signing CA.

Result: malicious pack not installed

—

Attack vector analysis

```
Archive:  Vendor.Pack.1.0.0.pack.signed
v0.8.1-alpha.1-18-gf0226dc:X:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Zip file size: 5774 bytes, number of entries: 12
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-17 17:55 dir/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file1
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file2
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir1/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file3
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file4
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir2/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir2/.gitignore
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir3/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir3/.gitignore
-rw-rw-r--  2.0 unx        0 bX defN 21-Jul-02 02:07 sample_file
-rw-rw-r--  2.0 unx      464 bX defN 21-Jul-13 02:49 Vendor.Pack.pdsc
12 files, 464 bytes uncompressed, 286 bytes compressed:  38.4%
```

Scenario: attacker invalidates or empties signature scheme

The signature follows a fixed format, and if it doesn't match it, it's recognized as invalid.

Note: vendors would have to signify in an external channel that the pack is signed (like the suggested .signed extension), to avoid a feature DoS.

Result: malicious pack not installed (if verification is enforced)

Attack vector analysis

```
Archive:  Vendor.Pack.1.0.0.pack.signed
v0.8.1-alpha.1-18-
gf0226dc:f LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUZpekNDQTNPZ0F3SUJBZ0lVQTBXcXhqSE5qZ2plUy9jVr
GDyHKqQN4mhuMq2+Keno38MN/z90x2R7kL7VXo/eqaKSj7MT4vqmbRTkITHXzQM7itnS/M//E4euQSsK3+ULcDi5ag5/
Labp9yV8MhxWYeklfDX8CiD5jplrWvv9BPjbkA48tC5WD6IKkmOkPvGnyqcXvh9p41VzUb8apWPYlWd2eBnuhHDqm0+NHcDr5
Nqv/
rxGmUWxDlrP2zOohdMMJUD6aYpL0GCyXT1SigUrBUHKHStxILAeC4stLldcdScgw1Q7FHlTUqLW8t5Z7WsuZI3E8CTVua1+b0
wNfJRfyJwqzpZJgKmPky/sxr24=
Zip file size: 5774 bytes, number of entries: 12
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-17 17:55 dir/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file1
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/file2
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir1/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file3
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir1/file4
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir2/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir2/.gitignore
drwxrwxr-x  2.0 unx        0 bx stor 22-Oct-13 10:19 dir/dir3/
-rw-rw-r--  2.0 unx        0 bX defN 22-Oct-13 10:19 dir/dir3/.gitignore
-rw-rw-r--  2.0 unx        0 bX defN 21-Jul-02 02:07 sample_file
-rw-rw-r--  2.0 unx      464 bX defN 21-Jul-13 02:49 Vendor.Pack.pdsc
12 files, 464 bytes uncompressed, 286 bytes compressed:  38.4%
```

Scenario: attacker modifies contents, signed hash and X.509 certificate, signed by a compromised trusted CA

If the pack is tampered and the attacker was able to compromise the CA signing the vendor's certificate, there's no way to differentiate from a legitimate one.

Result: malicious pack installed

Pros and Cons - Pros

Self contained

Does not change the Open-CMSIS-Pack's pack specification in any way (i.e the schema used in public indexes or the PDSC files).

Local / offline use

Same usage/guarantees offline as installing from an online public index/repository.

Varying degrees of complexity

From full PKI infrastructure to PGP keys, the signee is free to choose the best method according to its needs.

Pros and Cons - Pros

Lightweight

Both signature creation and checking are very light to compute, as SHA256 algo and PKCS1v15 padding are optimized in modern systems.

Compatible and interchangeable protocol

Other Open-CMSIS-Pack tools can use this protocol interchangeably with cpackget, just need to standardize their own version tags (making sure the tools accepts them).

CI friendly

All needed inputs are configured through flags, which easily integrates with any CI system.

Pros and Cons - Pros

Simple to develop/debug

The signature scheme is made with simplicity in mind. A Bash script could verify a signed pack in the same way as cpackget does (assuming it has the right tools like zipinfo, grep, base64, etc..) and it's human-readable.

Pros and Cons - Cons

Zip size limitation

.zip files are defined to have a maximum general comment field of 64KiB. The signed hash is of constant size, and unless the X.509 certificate is using a complex, multi signature scheme or huge certificate chain, this is not a relevant problem (dependends on the decided chain of trust).

Possible denial of service

As shown in slide 16, an attacker can empty the signature, rendering the pack as “unsigned”. For online usage, a standard must be defined to signal a pack as signed.

Manual certificate verification

As it stands, cpackget only performs some basic validations (like expiration date) on the X.509 certificate. Some technical knowledge is needed to confirm that it is OK.

Future work and Requirements

Trusted CA

An entity must be designed as the trusted CA, that emits and signs certificates for the vendors to use. This entity is also responsible for a) handing to the publishers the certificate/keys or b) sign the packs themselves, acting as a “middleman” for the vendors.

Establishing an accepted certificate chain

The trusted CA can act as a Root CA for all vendors, or be an intermediate CA that is signed by a non Open-CMSIS-Pack entity (like *Let’s Encrypt*). This trusted CA could emit vendor certificates so vendors could emit their “sub” certificates, but a consensus in the protocol must be reached.

User side certificate handling

On the user end, should cpackget ship with the Trusted CA’s certificate embed in the binary? Should the user manually install it from a trusted source? Should vendor certificates be stored in the OS keychain/keyring?

Thank you. Questions?