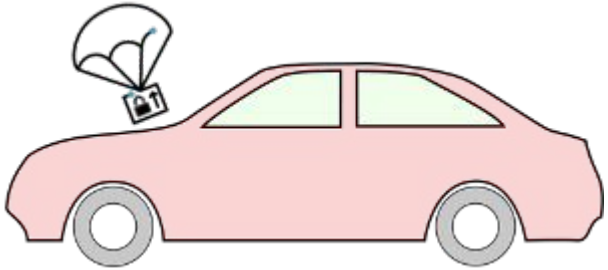
Uptane

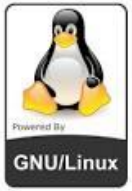
Securing Over-the-Air Updates Against Nation State Actors



Justin Cappos
New York University

NYU TANDON SCHOOL OF ENGINEERING

What do these companies have in common?



What do these companies have in common?




Users attacked via software updater!

Windows

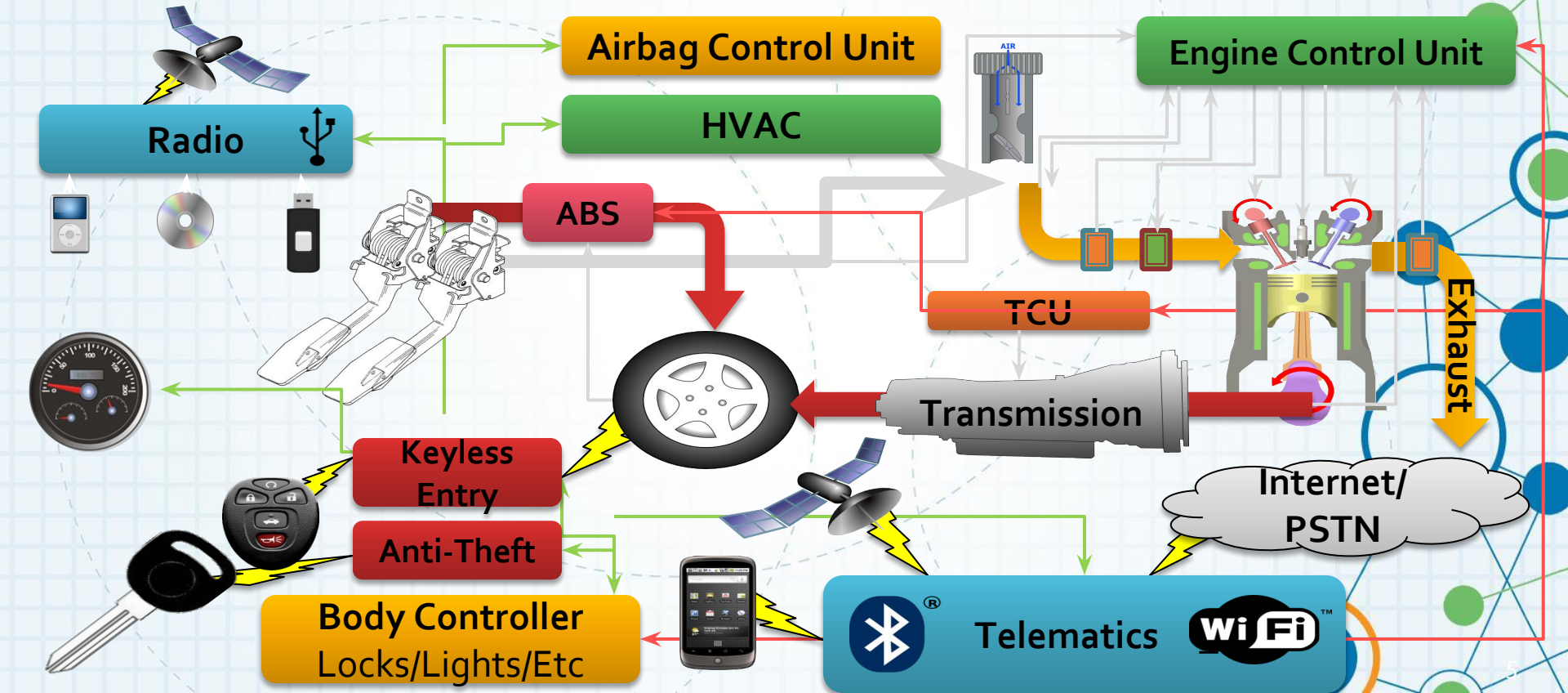
sourceforge

GitHub

Software repository compromise impact

- SourceForge mirror distributed **malware**.
- Attackers impersonate Microsoft Windows Update to spread Flame **malware**.
- Attacks on software updaters have massive impact
  - E.g. South Korea faced 765 million dollars in damages.
- NotPetya spread via software updates!

The modern automobile



Cars Are Dangerous

- Researchers have made some scary attacks against vehicles
  - remotely controlling a car's brakes and steering while it's driving
  - spontaneously applying the parking brake at speed
  - turning off the transmission
  - locking driver in the car

Cars are multi-ton, fast-moving weapons

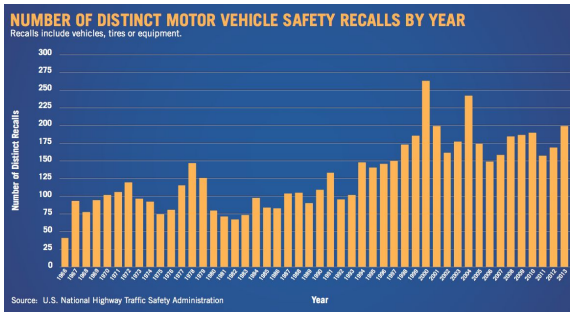
People will die

Updates Are Inevitable

- Millions of lines of code means bugs
- Regulations change -> firmware must change
- Maps change
- Add new features
- Close security holes
- Cars move across borders…

Updates Must Be Practical

- Updating software/firmware has often meant recalls.
- Recalls are extremely expensive
  - GM spent $4.1 billion on recalls in 2014
  - GM's net income for 2014 was < $4 billion
  - People do not like recalls.
- Updates must be over the air.



Updates Are Dangerous

- Update -> Control

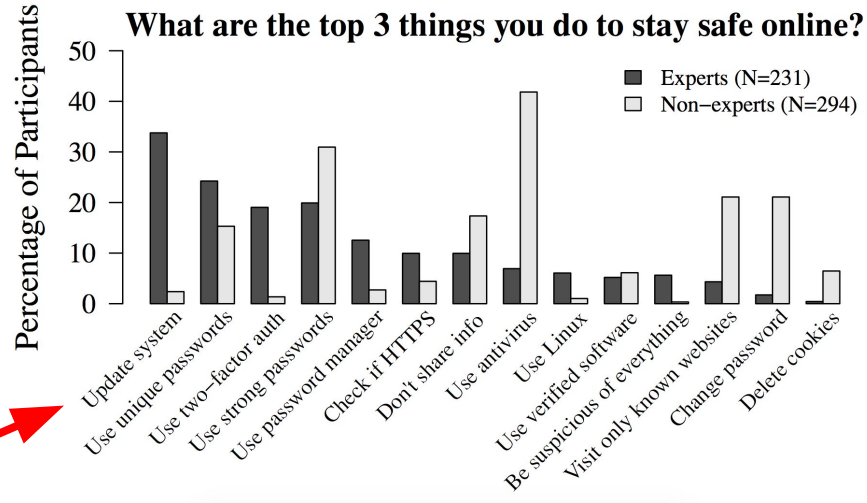
Secure Updates

- Nation-state actors pull off complex attacks
  - Must not have a single point of failure

What to do?

Must update to fix security issues

Insecure update mechanism is a new security problem

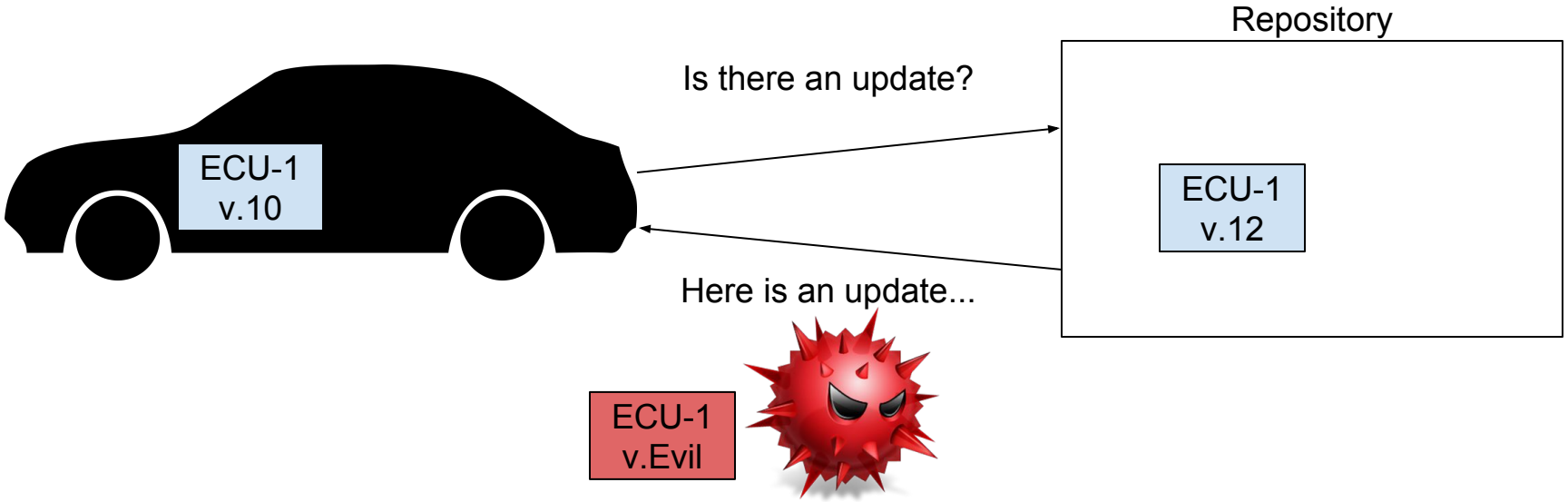


"...No one Can Hack My Mind": Comparing Expert and Non-Expert Security Practices
Ion, et al. SOUPS 2015

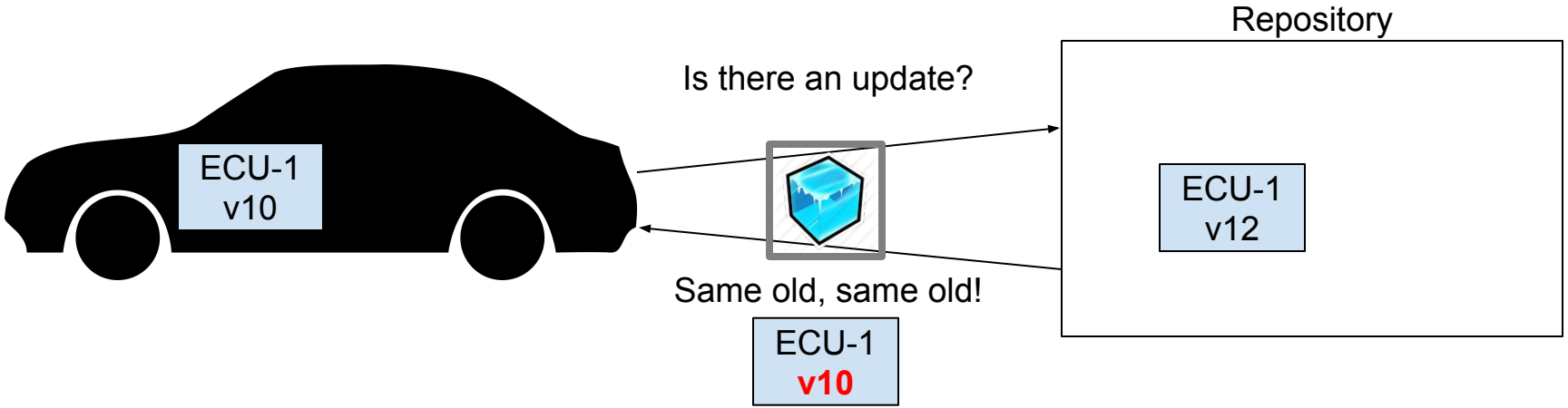
Attacks

What are some of the attacks?

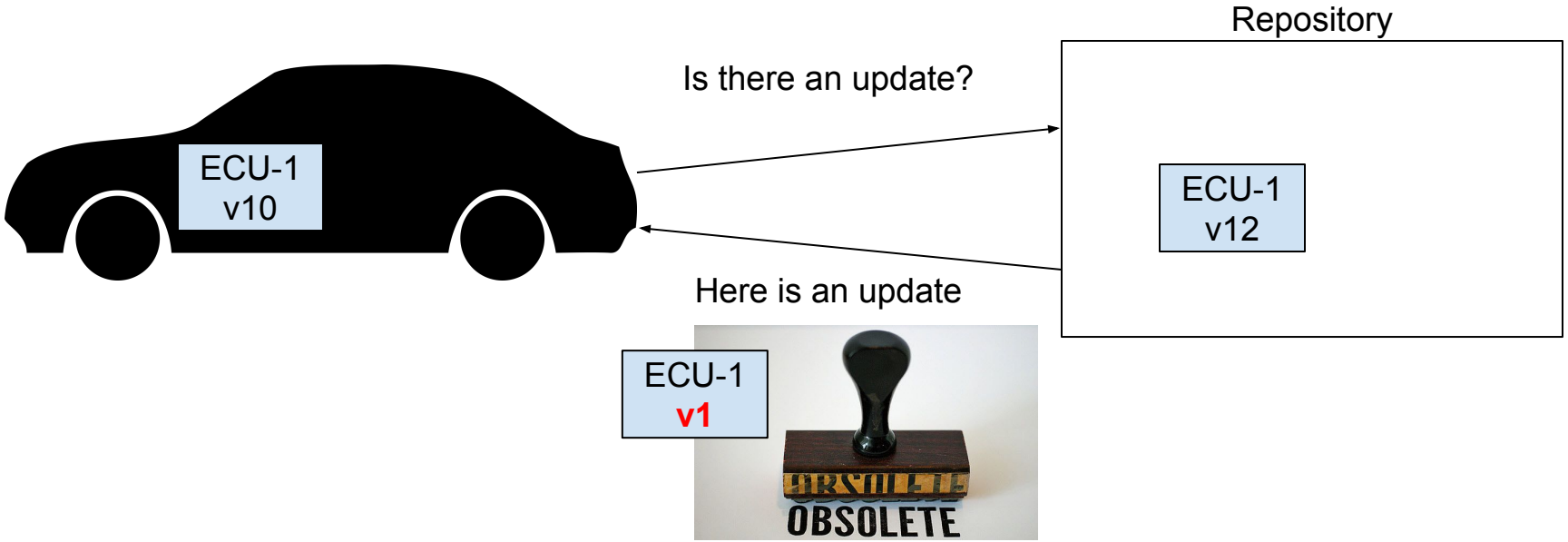
Arbitrary software attack



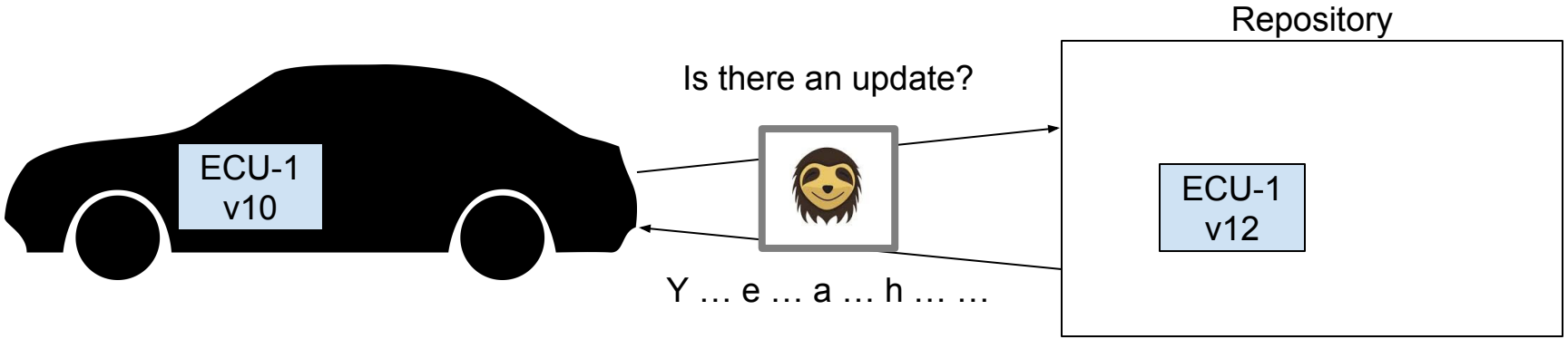
Freeze attack



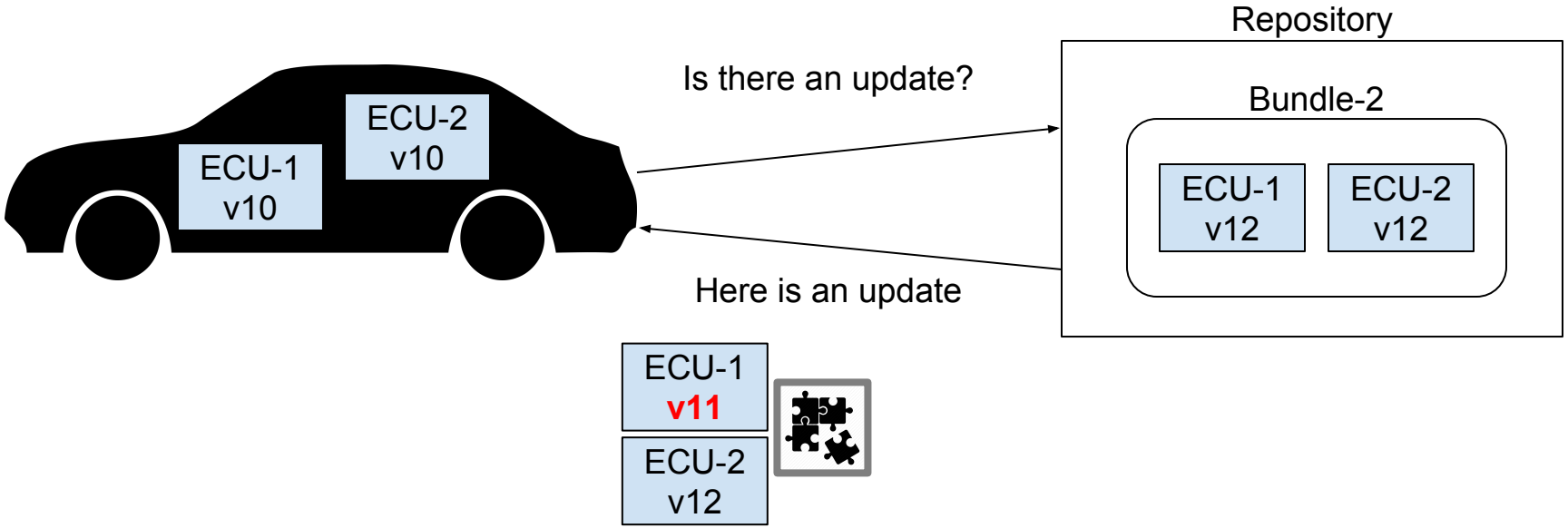
Rollback attack



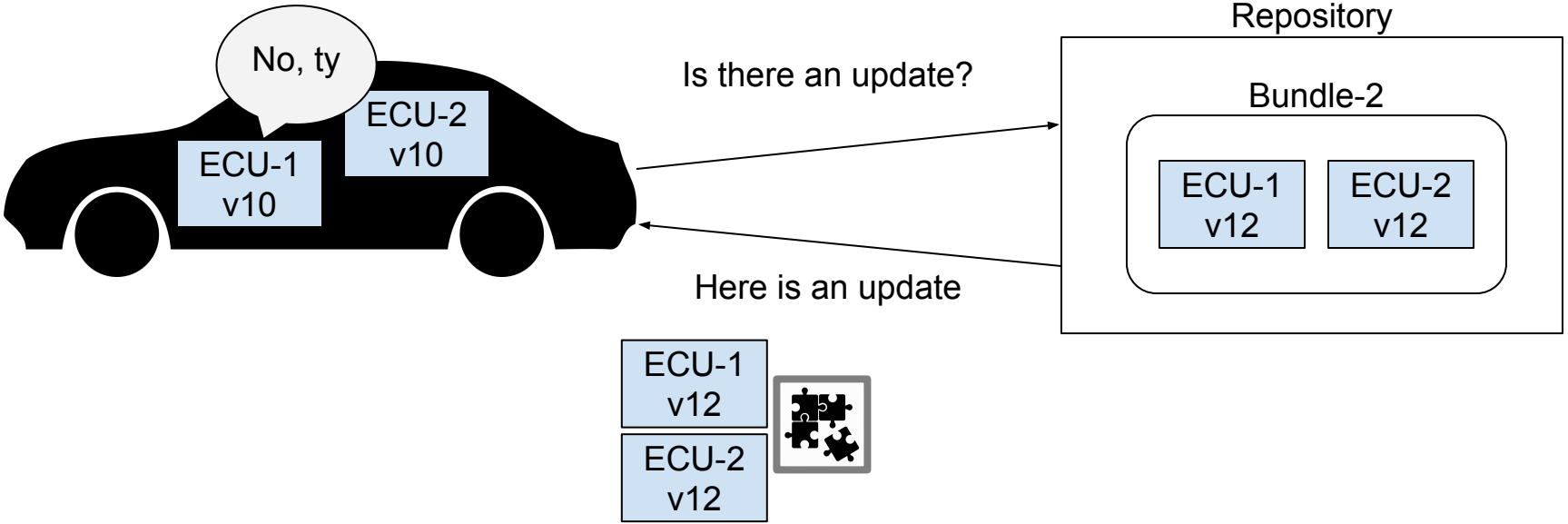
Slow retrieval attack



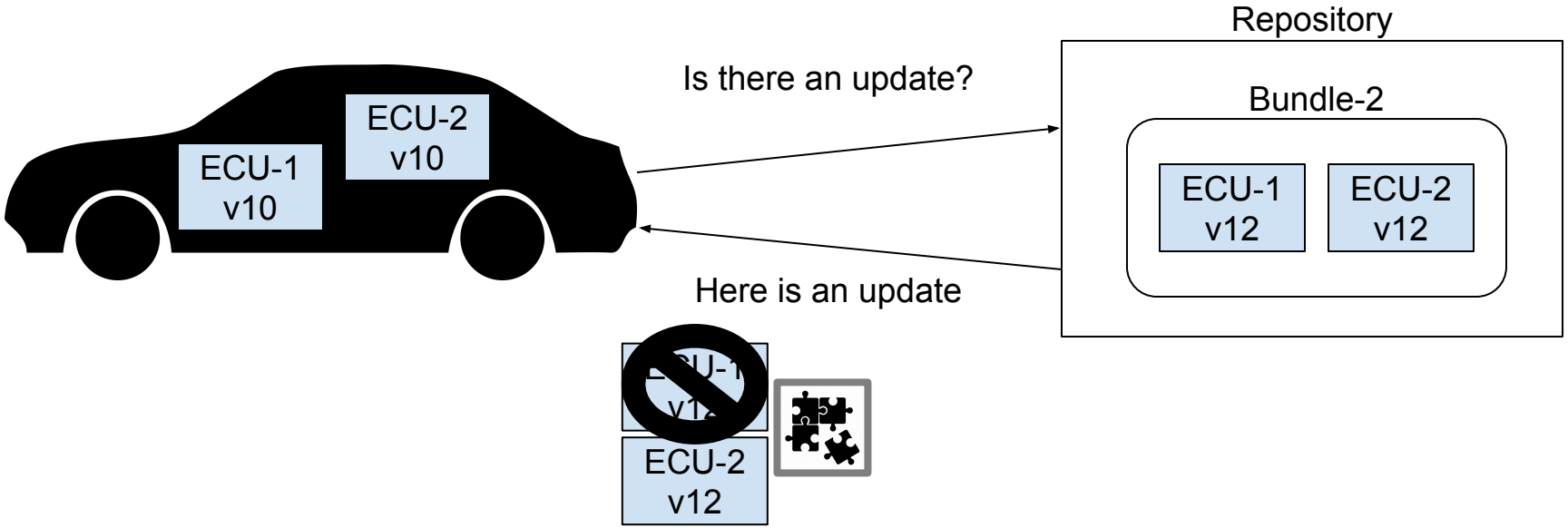
Mix and Match attacks



Partial Bundle attack

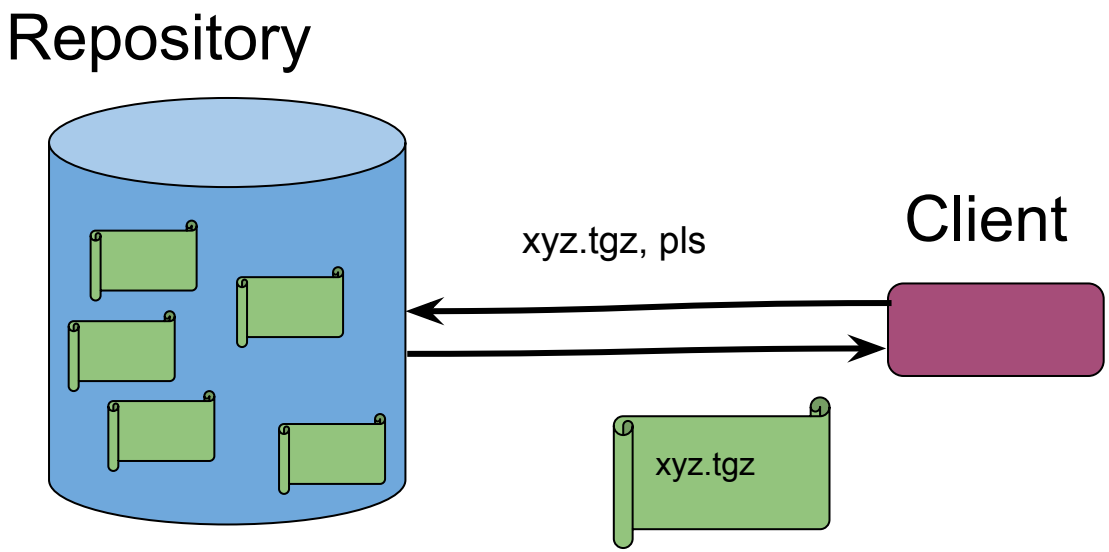


Partial Freeze attack



So how do people try to prevent these attacks?

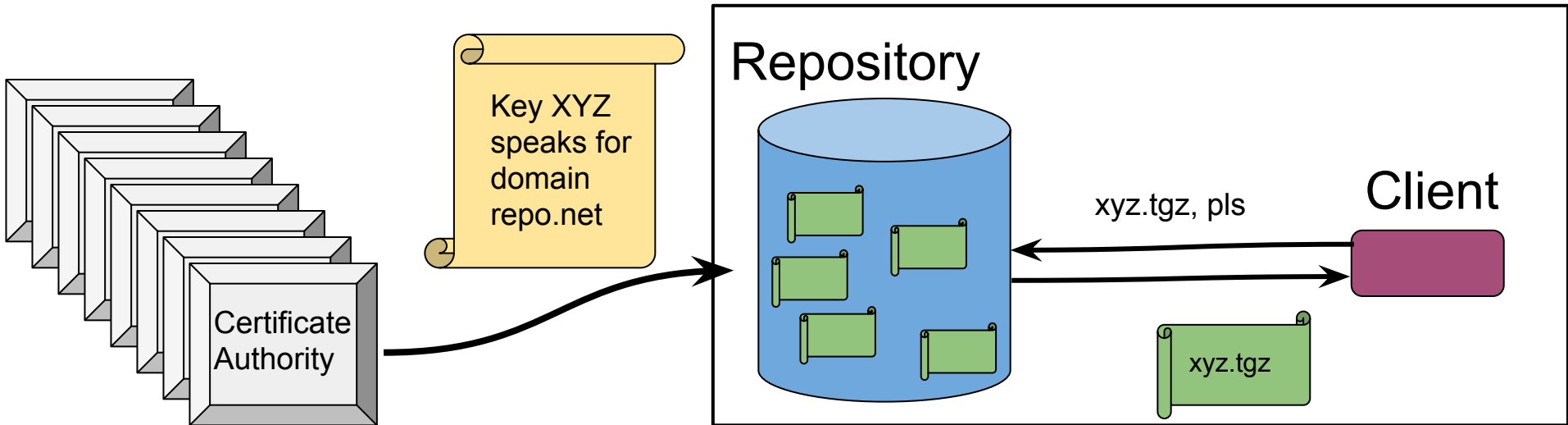
Update Basics



Inadequate Update Security 1: TLS/SSL

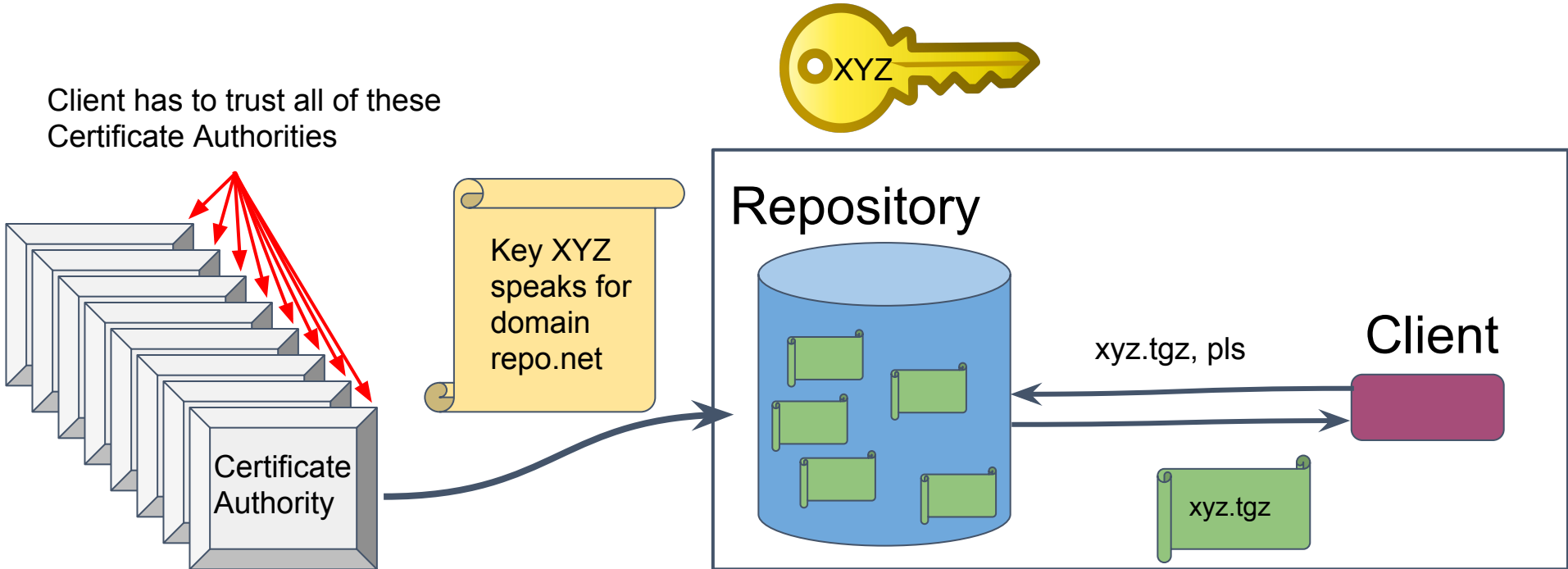
Traditional solution 1:

Authenticate the repository (TLS, SSL, etc)



Inadequate Update Security 2: TLS/SSL

Transport Layer Security: Problem 1

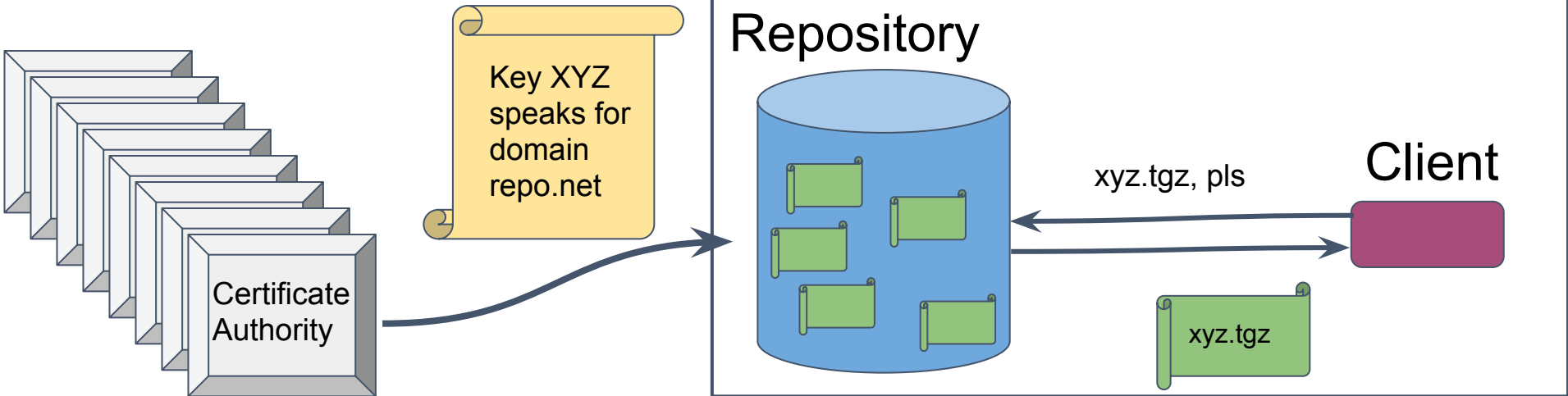


Inadequate Update Security 3: TLS/SSL

Transport Layer Security: Problem 2

Client has to trust this key.

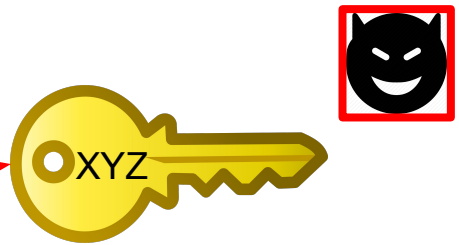
… which HAS to exist ON the repository, to sign communications continuously.



Inadequate Update Security 4: Just Sign!

Traditional Solution 2:

Sign your update package with a specific key.
Updater ships with corresponding public key.

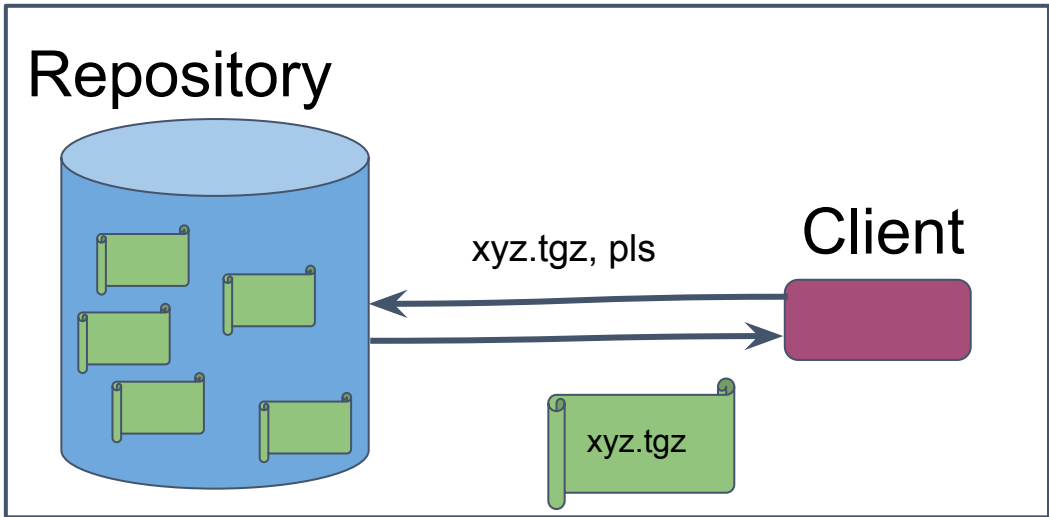


Client has to trust this key

… used for every update to the repository.

… key ends up on repo or build farm.

If an attacker gains the use of this key, they can install arbitrary code on any client.

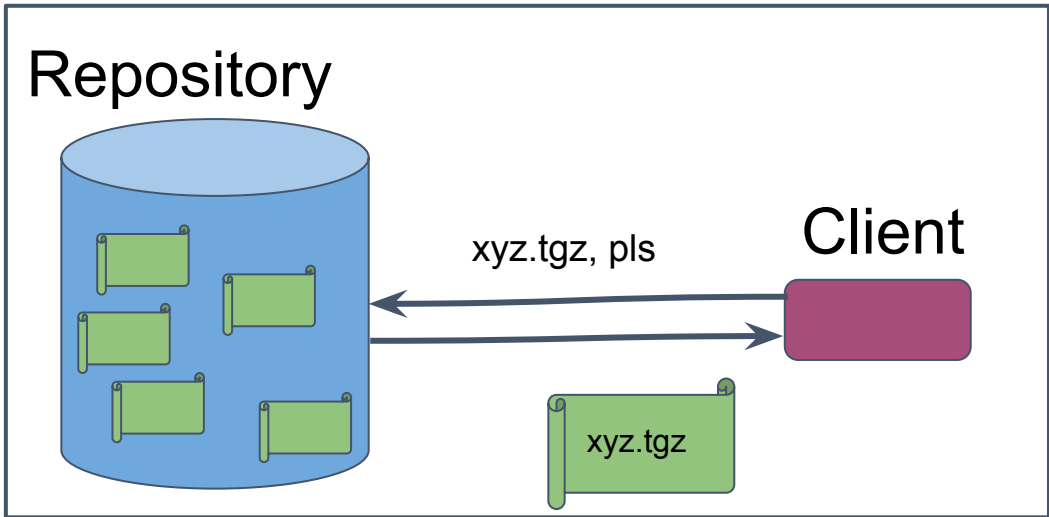


Update Security

We need:

- To survive server compromise with the minimum possible damage.
  - Avoid arbitrary package attacks
- Minimize damage of a single key being exposed
- Be able to revoke keys, maintaining trust
- Guarantee freshness to avoid freeze attacks
- Prevent mix and match attacks
- Prevent rollback attacks
- Prevent slow retrieval attacks
- ...

Must not have single point of failure!



The Update Framework (TUF)

Linux Foundation CNCF project

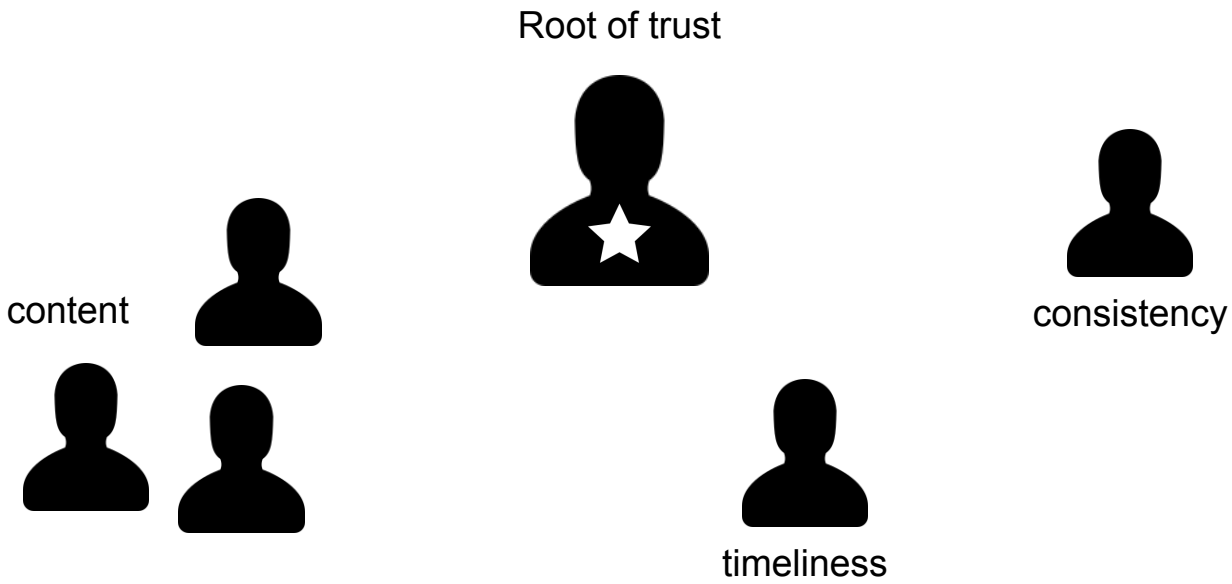
CII Best Practices Silver Badge

TUF goal “Compromise Resilience”

- TUF secures software update files
- TUF emerges from a serious threat model:
  - We do NOT assume that your servers are perfectly secure
  - Servers will be compromised
  - Keys will be stolen or used by attackers
  - TUF tries to minimize the impact of every compromise

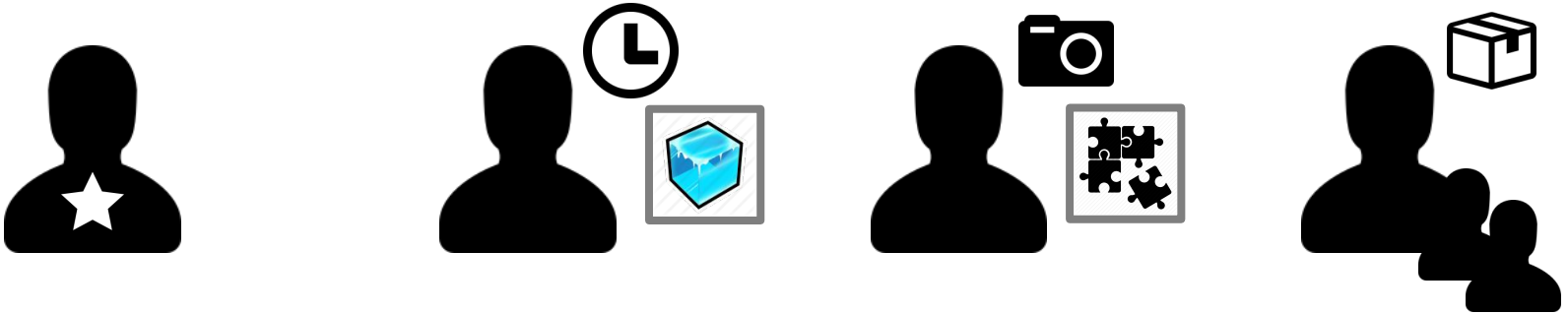
The Update Framework (TUF)

Responsibility Separation



The Update Framework (TUF)

TUF Roles Overview



Root

(root of trust)

Timestamps

(timeliness)

Snapshot

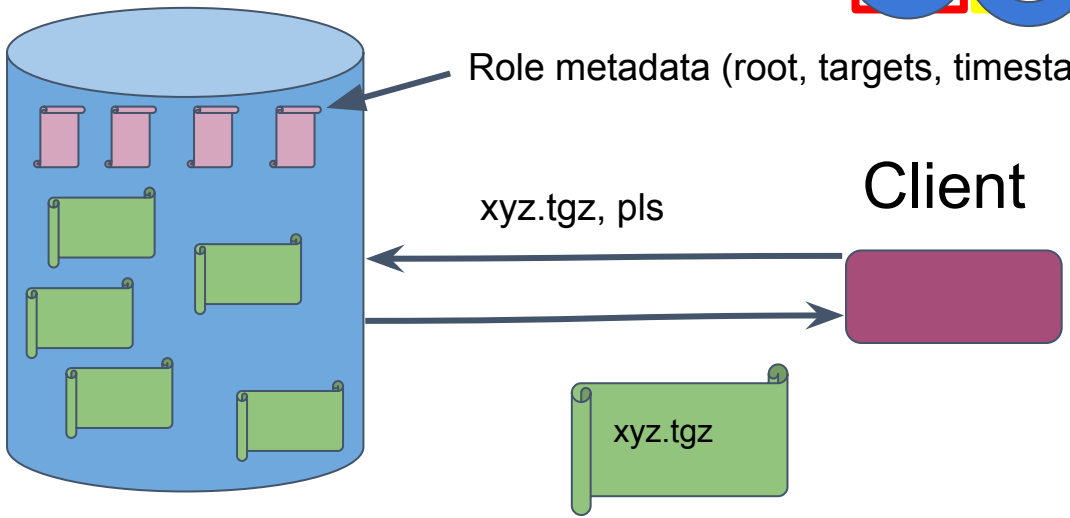
(consistency)

Targets

(integrity)

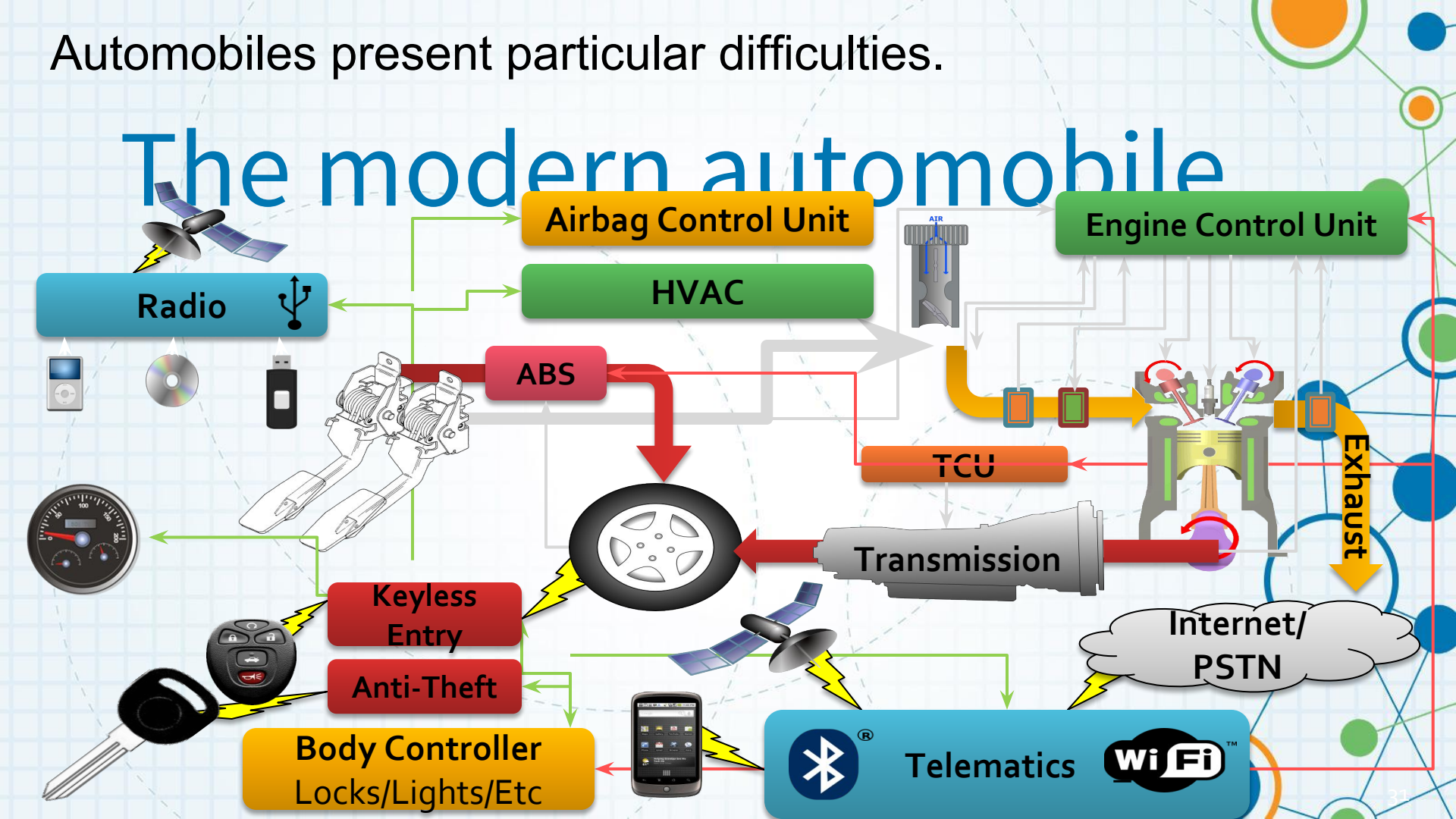
The Update Framework (TUF)

Repository



Automobiles present particular difficulties.

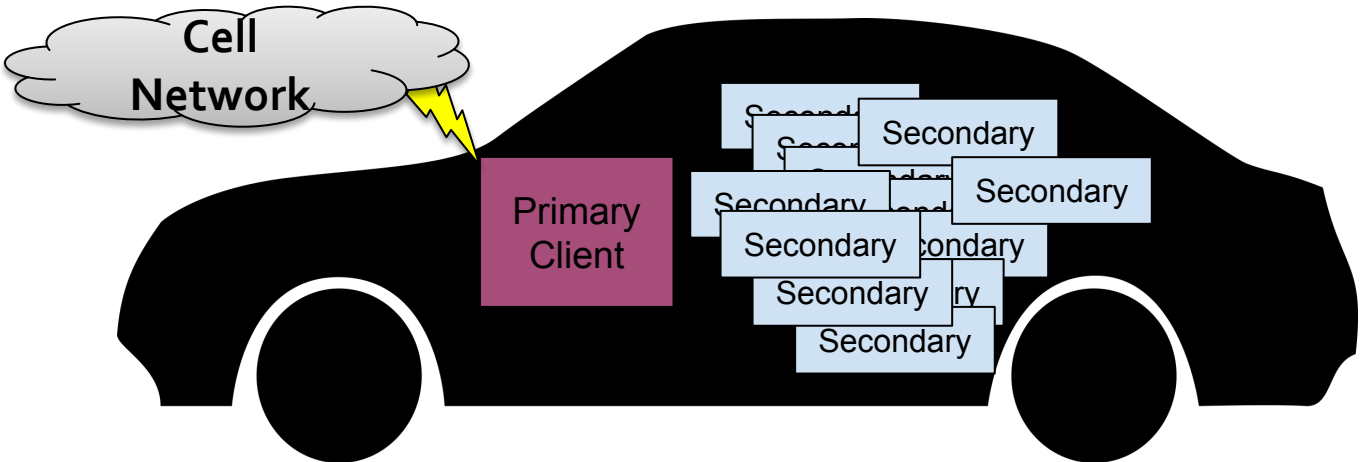
The modern automobile



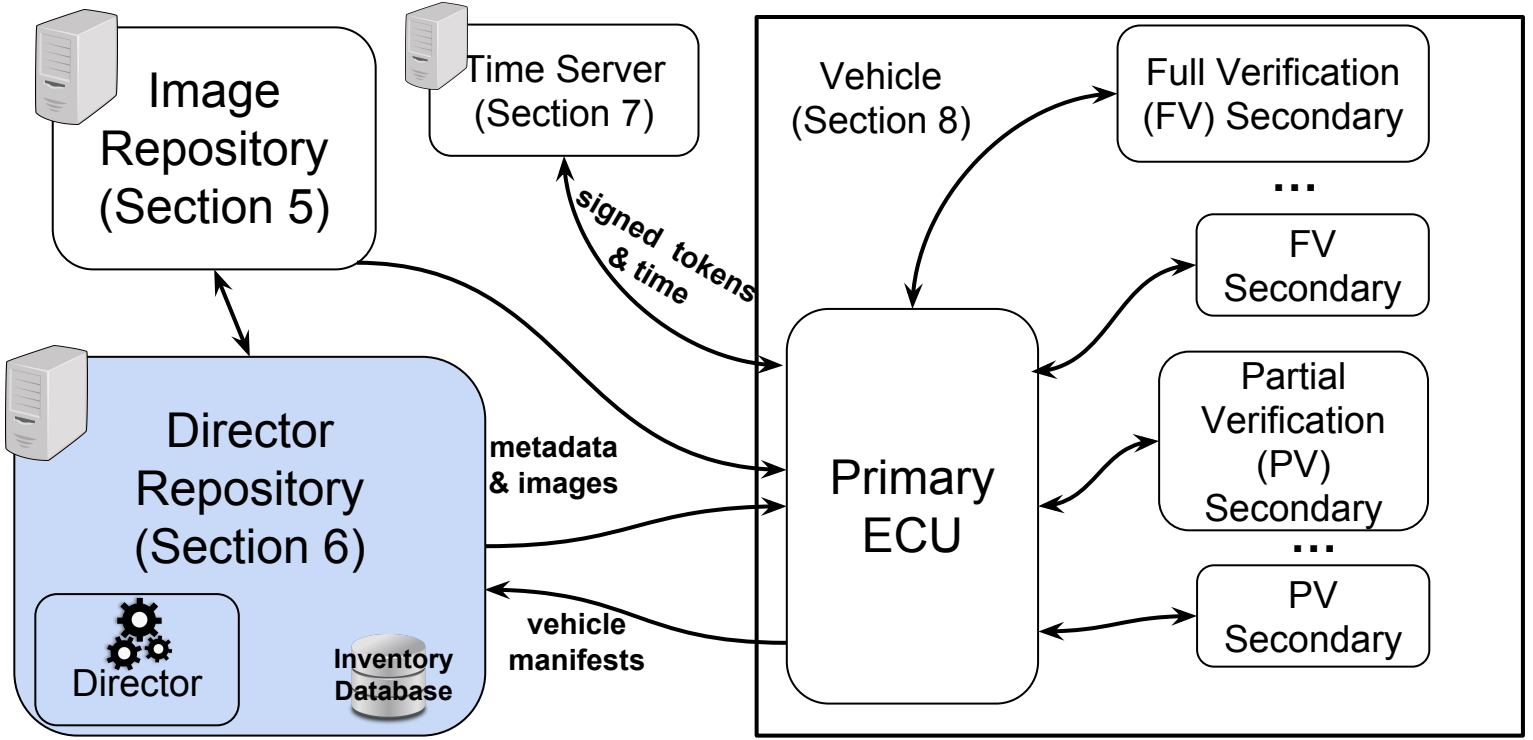
Uptane builds on The Update Framework (TUF)

- Timeserver
- Multiple Repositories: Director and Image Repository
- Manifests
- Primary and Secondary clients
- Full and Partial verification

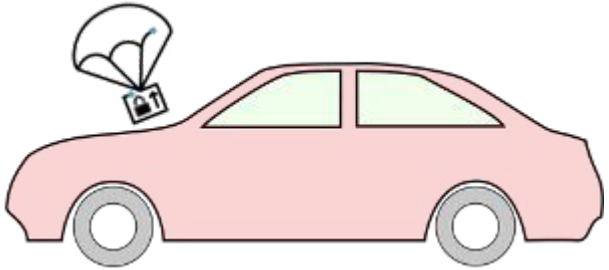
Uptane: Client-side Basics



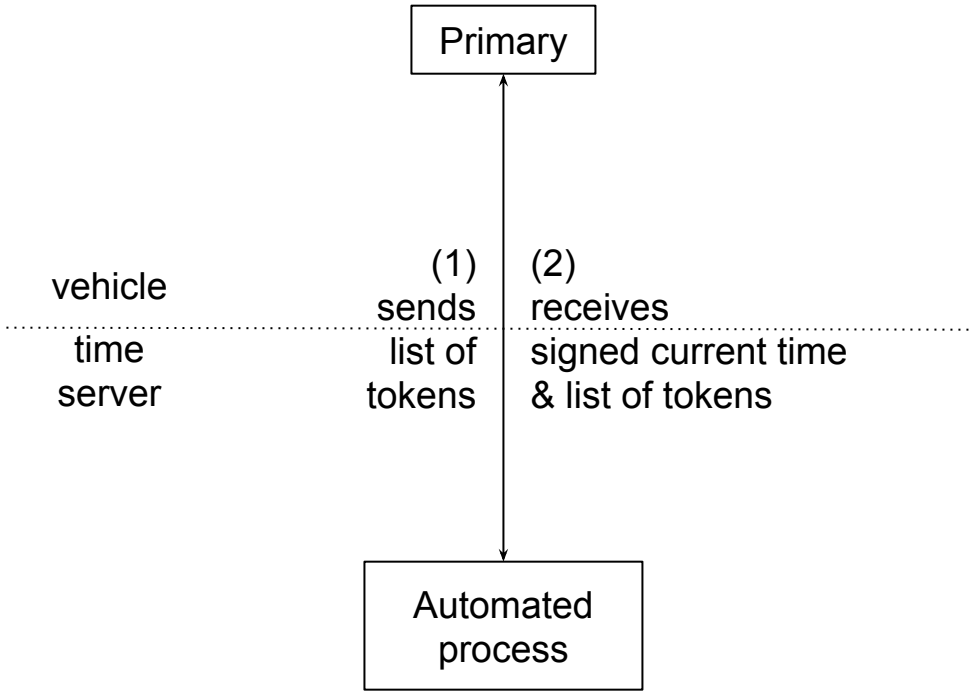
Uptane: High level view



Time server

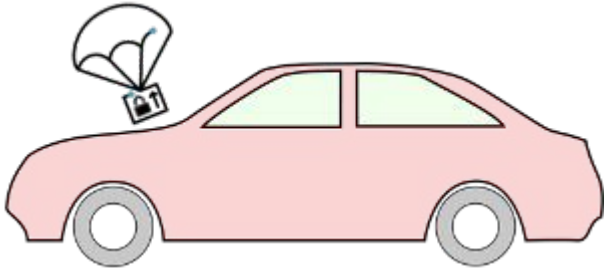


Time server

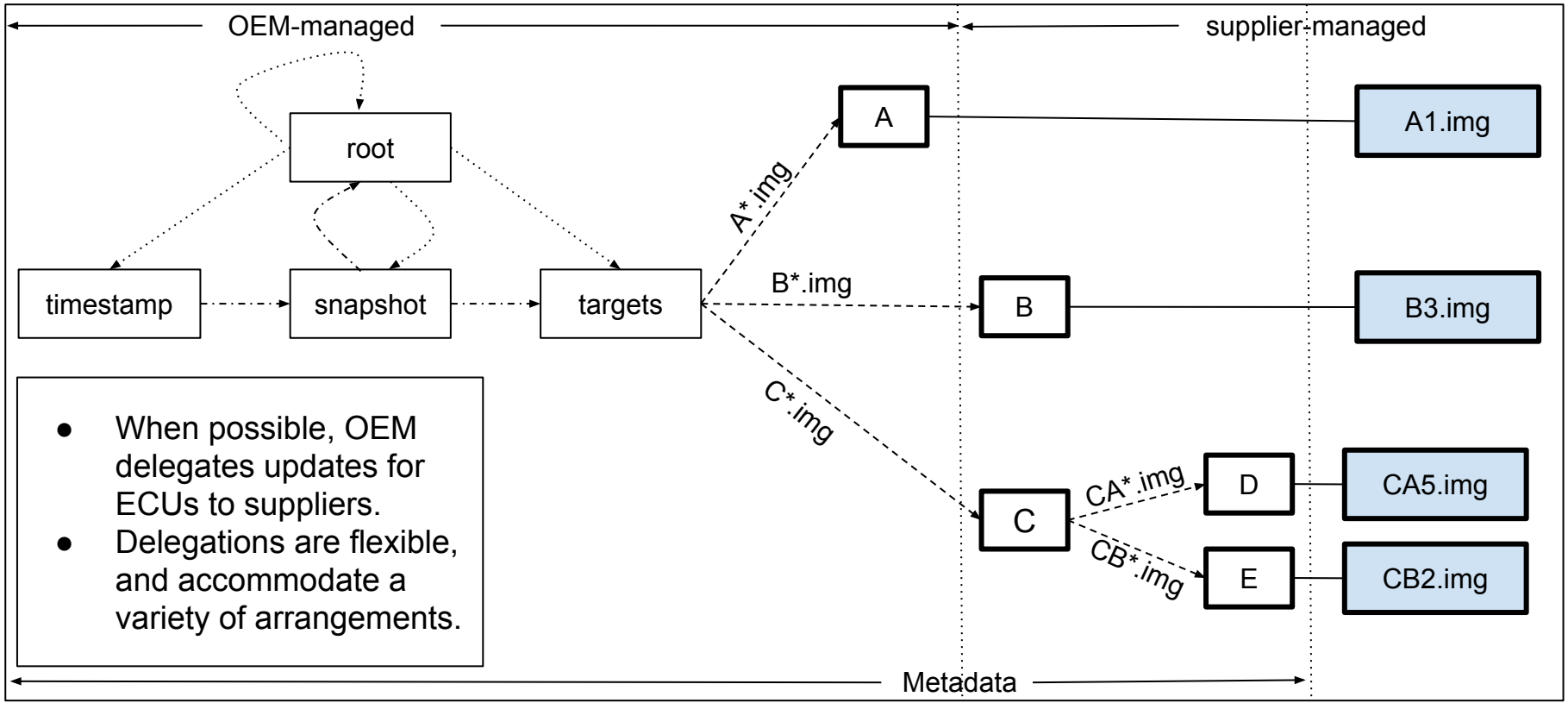


- A primary sends a list of tokens, one for each ECU, to the time server.
- An automated process on the time server returns a signed message containing: (1) the list of tokens, and (2) the current time.

Image repository

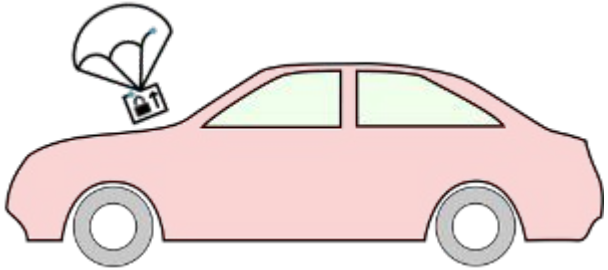


The image repository



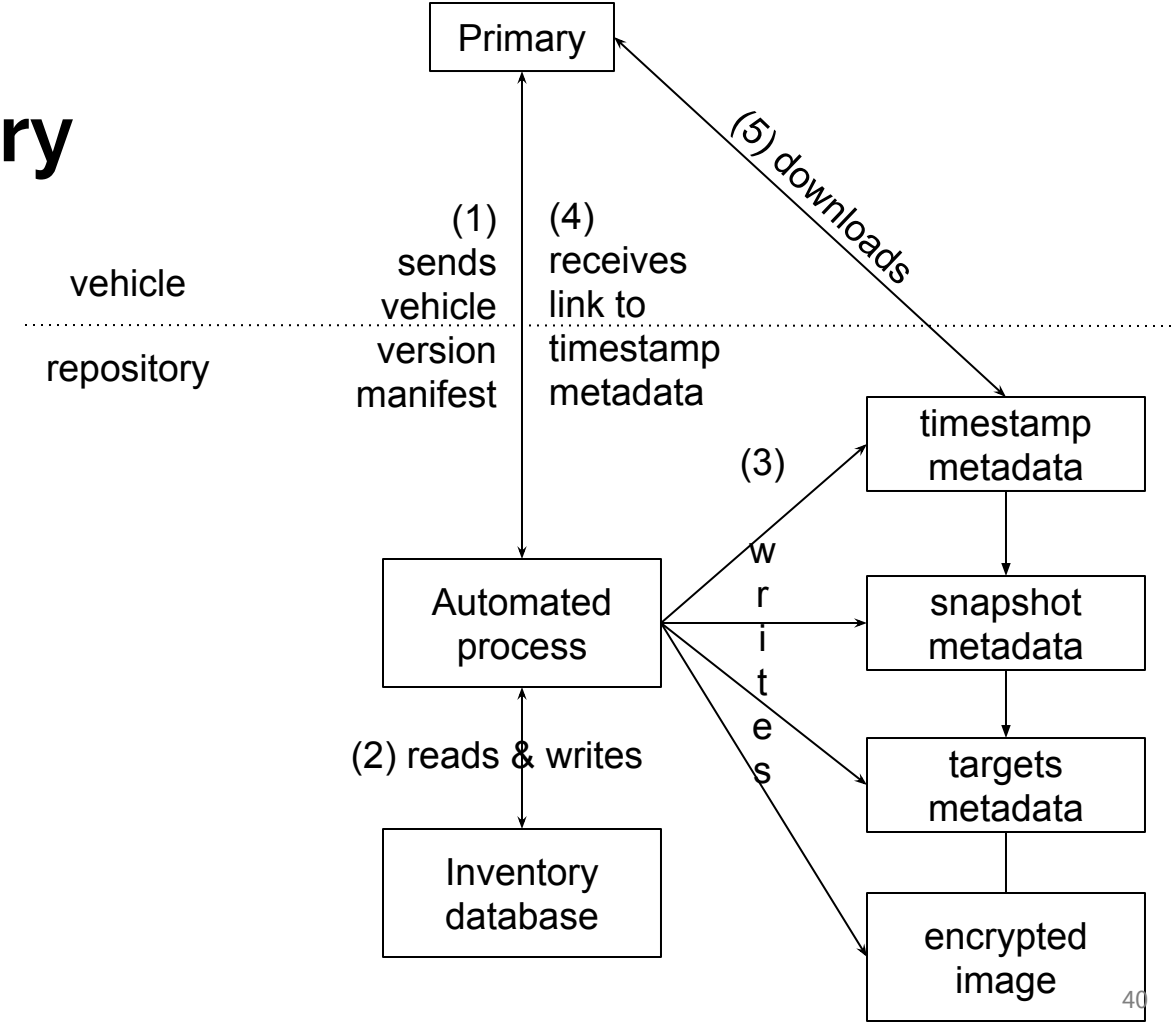
- When possible, OEM delegates updates for ECUs to suppliers.
- Delegations are flexible, and accommodate a variety of arrangements.

Director repository



Director repository

- Records vehicle version manifests.
- Determines which ECUs install which images.
- Produces different metadata for different vehicles.
- May encrypt images per ECU.
- Has access to an inventory database.



Big picture

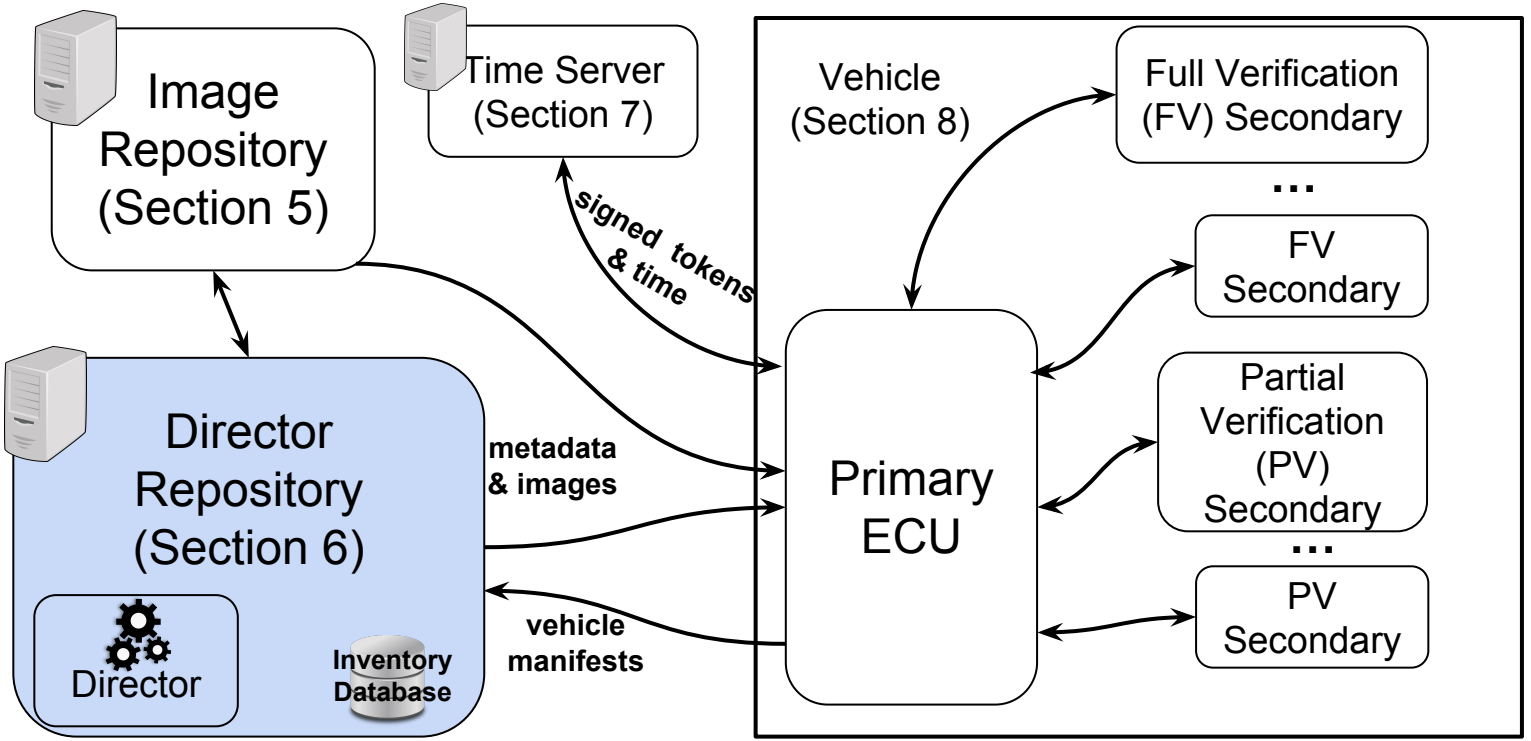
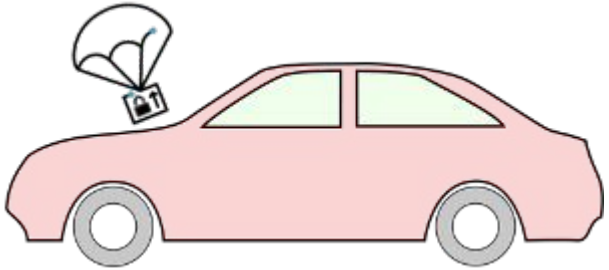


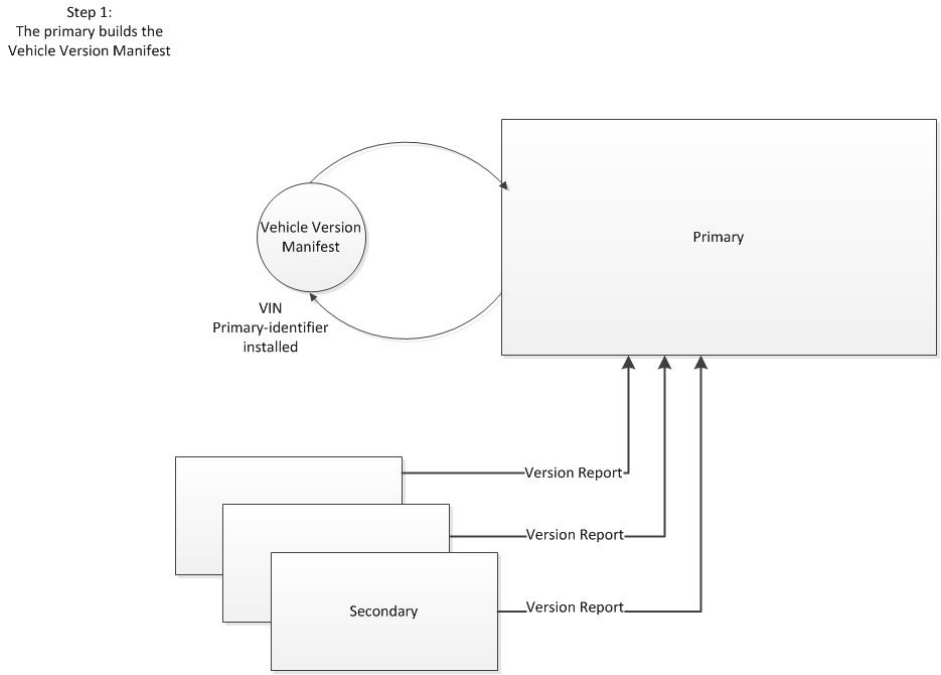
Image Repository (Section 5)
Time Server (Section 7)
signed tokens & time
Vehicle (Section 8)
Full Verification (FV) Secondary
...
FV Secondary
Partial Verification (PV) Secondary
...
PV Secondary
Primary ECU
Director Repository (Section 6)
metadata & images
vehicle manifests
Director
Inventory Database

Uptane workflow on vehicle



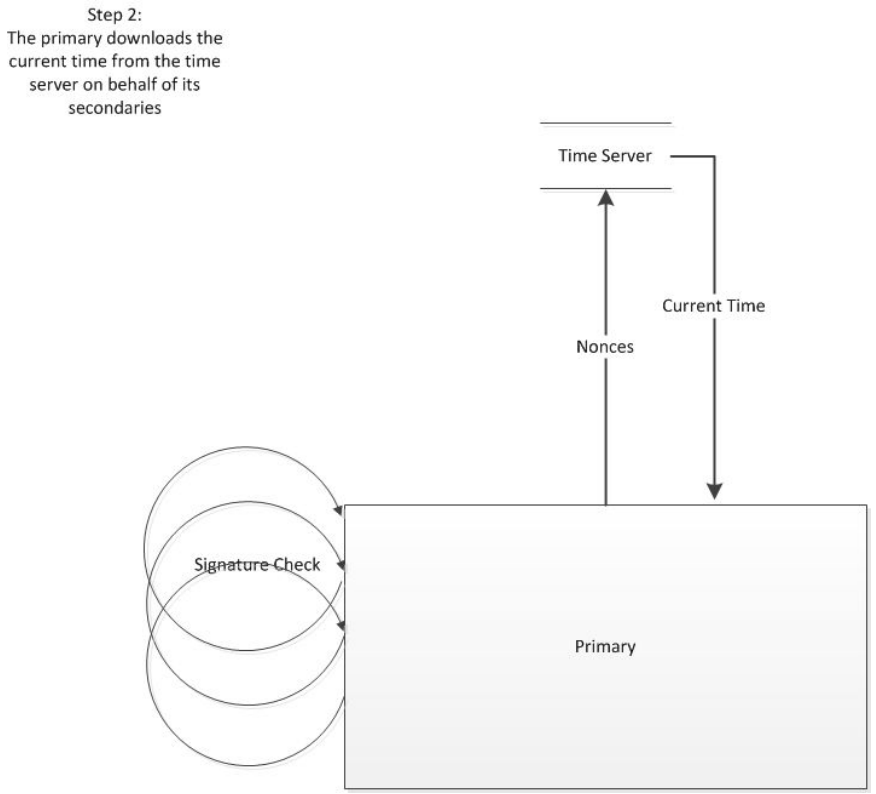
Downloading updates (1)

- Primary receives an ECU Version Manifest and a nonce from each Secondary.
- Primary produces Vehicle Version Manifest, a signed record of what is installed on Secondaries
- Primary sends VVM to Director
- Primary sends nonces to Timeserver



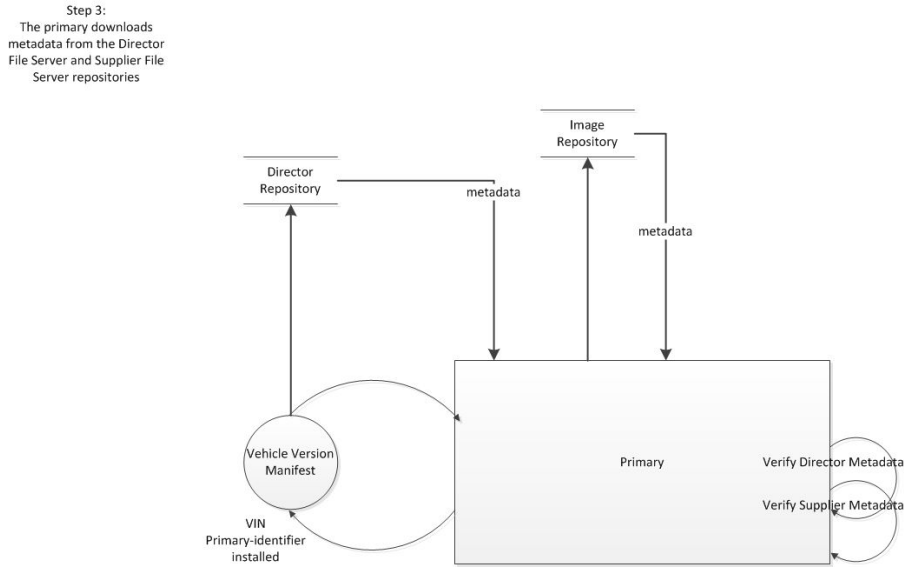
Downloading updates (2)

- Timeserver returns the signed [time and nonces] to the Primary.



Downloading updates (3)

- The primary downloads metadata from both the Director and Image repositories on behalf of all ECUs
- The primary performs *full verification* of metadata on behalf of all secondaries.



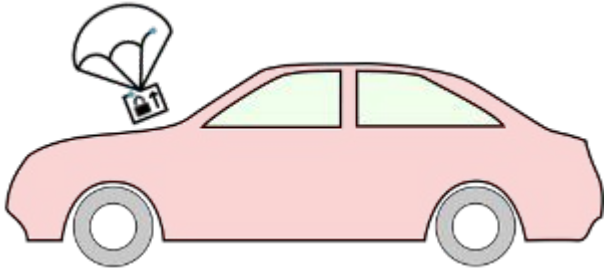
Full verification

1. **Load the latest downloaded time from the time server.**
2. **Verify metadata from the director repository.**
    a. Check the root metadata file.
    b. Check the timestamp metadata file.
    c. Check the snapshot metadata file.
    d. Check the targets metadata file.
3. **Download and verify metadata from the image repository.**
    a. Check the root metadata file.
    b. Check the timestamp metadata file.
    c. Check the snapshot metadata file, especially for rollback attacks.
    d. Check the targets metadata file.
    e. For every image A in the director targets metadata file, perform a preorder depth-first search for the same image B in the targets metadata from the image repository, and check that A = B.
4. **Return an error code indicating a security attack, if any.**

Partial verification

1. **Load the latest downloaded time from the time server.**
2. **Load the latest top-level targets metadata file from the director repository.**
    a. Check for an arbitrary software attack. This metadata file must have been signed by a threshold of keys specified in the previous root metadata file.
    b. Check for a rollback attack.
    c. Check for a freeze attack. The latest downloaded time should be < the expiration timestamp in this metadata file.
    d. Check that there are no delegations.
    e. Check that every ECU identifier has been represented at most once.
3. **Return an error code indicating a security attack, if any.**

Uptane status / wrap up



Uptane “Reference” Implementation

- Goal: Assist other implementers
  - Code readability is a primary goal

- Not the most popular implementation in practice (by design)
  - Readability > performance / implementation size
    - Most TUF deployments do not use the reference implementation
  - Useful as a reference, conformance testing, etc.

- Open source, free to use (MIT License)
  - Other groups are free to contribute!

Security Reviews

Reviews of implementations and design:

- Cure53 audited ATS's Uptane implementation
- NCC Group audited Uptane's reference implementation (pre-TUF fork)
- SWRI finalizing Uptane reference implementation / specification audit
- ...

Uptane Integration

Work closely with vendors, OEMs, etc.

- Security reps from 78% of cars
- Many top suppliers / vendors
    - ~12-35% of cars on US roads
- Automotive Grade Linux
- OEM integrations
    - Easy to integrate!

AUTOMOTIVE GRADE LINUX

Press

- Dozens of articles
- TV / Radio / Newspapers / Magazines

POPULAR SCIENCE

WANT MORE?

TECHNOLOGY

The year's most important innovations in security

A botnet vaccine, a harder drive, and 3-D bag scanner.

By Kelsey D. Atherton and Rachel Feltman October 17, 2017

This article is a segment of 2017's Best of What's New list. For the complete tabulation of the year's most transformative products and discoveries, head right this way.

Year By

lligence Group BIG
d data
ard companies,
ce.

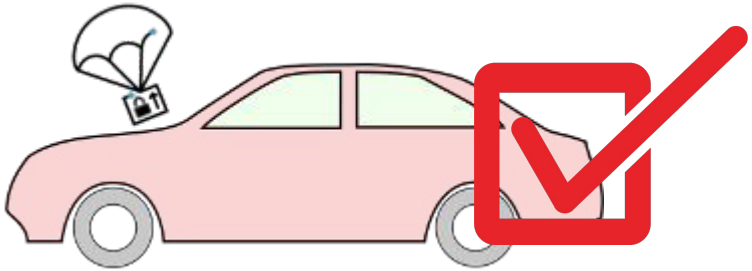
paigns-at scale-
with highly refined vehicle and device targeting, discrete policy and privacy controls, fully customizable consumer communications,
and solution deployment flexibility. In addition to the features announced in early 2017, OTAmatic now includes:

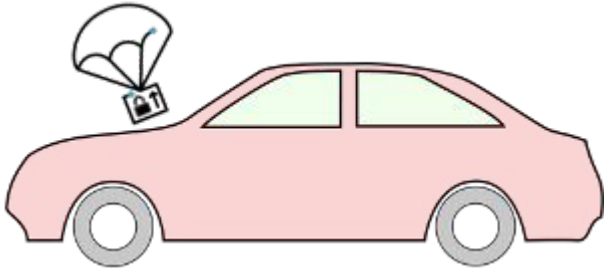
Get Involved With Uptane!

- Workshops
- Technology demonstration
- Compliance tests
- Standardization (IEEE / ISTO)
- Join our community! (email: jcappos@nyu.edu or go to the Uptane forum)

https://uptane.github.io/

Homeland Security





For more details, please see the Implementation Specification and other documentation at **uptane.github.io**

Cars are heavily computerized

- Today's car is a big distributed system
  - Complex computerized control
    - Millions of lines of code
    - ~100 distinct computers (**ECU**s: Electrical Control Units)
    - Average car last year had about 80
    - Some luxury or hybrid cars last year had around 150
  - Shared internal networks (CAN, FlexRay, Ethernet, …)
  - Increasing external comm. features
    - Telematics, Bluetooth, TPMS, RDS, XM radio, GPS, keyless start/entry, USB ports, WiFi, etc
- Tomorrow's car -> much more of everything
  - traffic control, autonomous driving, … jetpacks?

In summary, cars are quickly becoming networks of embedded systems with multiple tons of attached mechanical parts that move around a bunch. I'm not a car person, so from my perspective, that is what a car is: four wheels and a whole lot of cheap computers with closed-source firmware, networked in a way that would make you cry.

Uptane: Software Update Security for Cars

Software updates

- Inevitable
- Dangerous

I hope you'll forgive me for having several slides to make what will in retrospect probably four very obvious points. But here we go.

((CLICK)) Software updates are necessary.

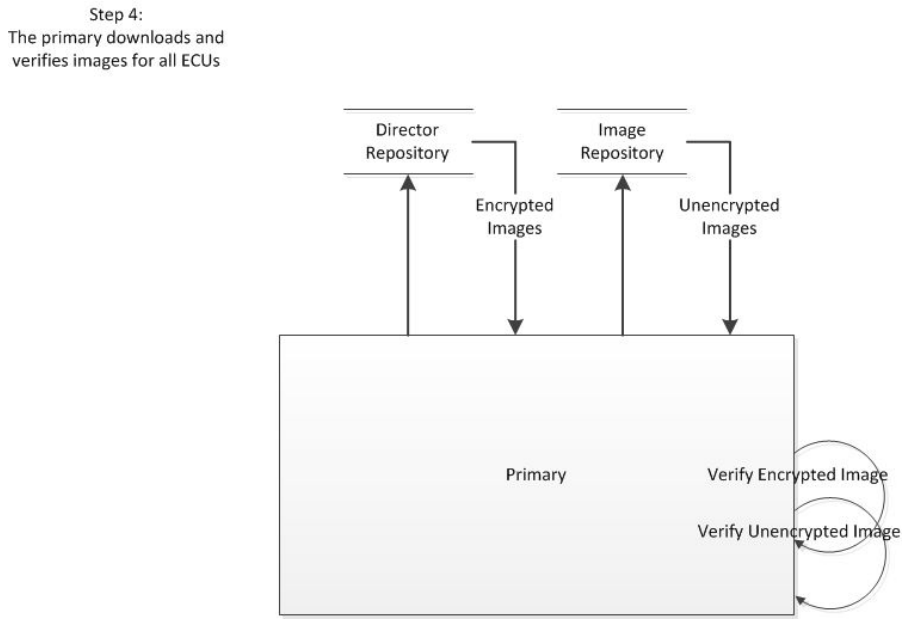
((CLICK)) Software updates are dangerous.

Cars Are Dangerous

- Cars are also multi-ton fast-moving weapons.
- Attacks by a nation-state actor could wreak havoc

Downloading updates (4)

- Encrypted images, if any, are downloaded from the director repository.
- Unencrypted images are downloaded from the image repository.



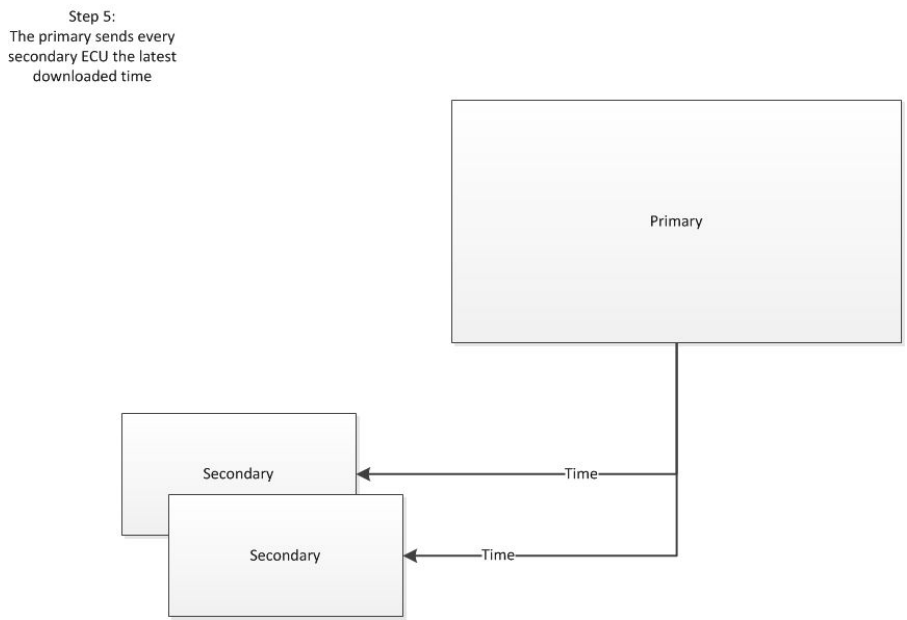
Downloading updates (5-7)

Primary distributes to Secondaries:

- Timeserver's time attestations
- Director and Image Repo metadata
- Update data for each Secondary

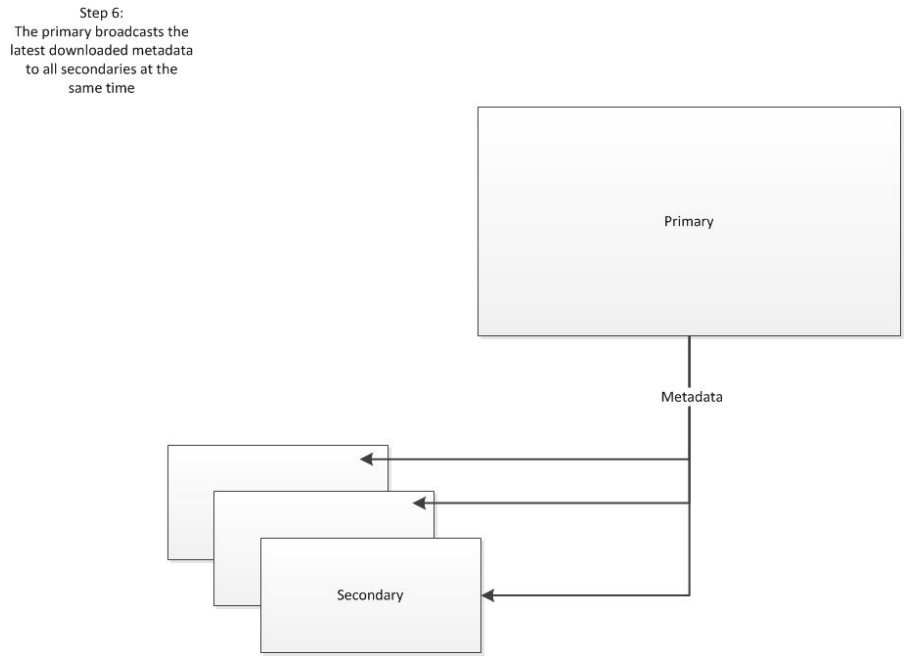
Downloading updates (5)

- The primary sends the timeserver's signed time to all of its secondaries.



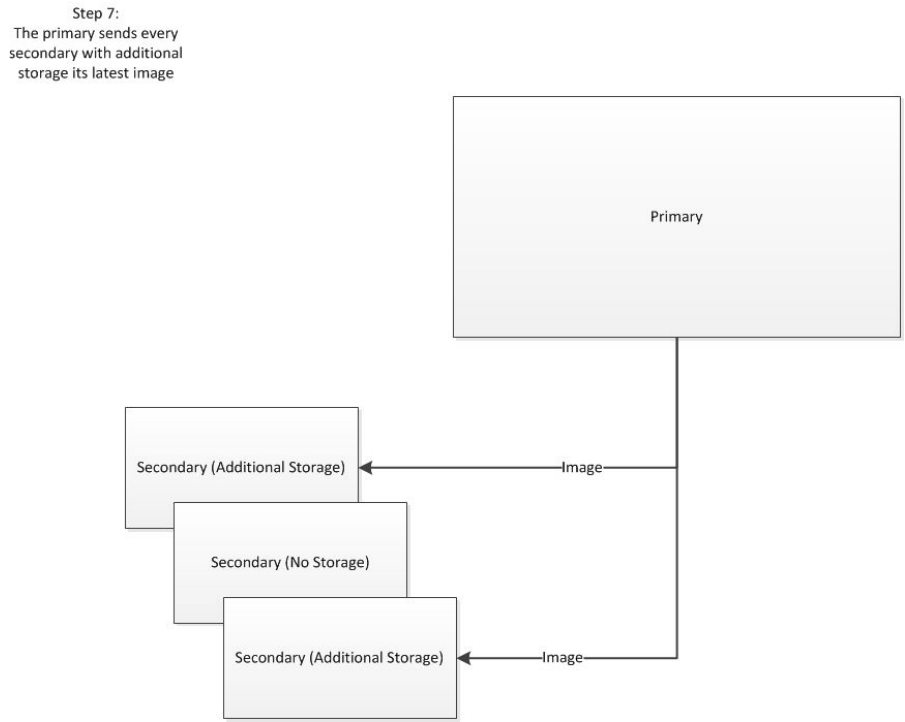
Downloading updates (6)

- The primary sends the latest downloaded metadata to all of its secondaries.



Downloading updates (7)

- Additional Storage
  (A/B firmware Storage)



Before Secondary installs an update (1)

1. **Verify the latest downloaded time.**
    a. Timeserver signature must be valid.
    b. List of nonces must include the nonce this Secondary sent in the last version report.
    c. The current time must be greater than the previous downloaded time.
    d. If all checks pass, then save the new time and generate a new token.
    e. Otherwise, reuse previous token.
2. **Verify metadata using full / partial verification.**
    a. (Discussed in more detail later.)
    b. Result is a trustworthy hash and file length for the image. That allows us to validate the image.
3. **If a secondary does not have additional storage, download image from primary.**
    a. May use primary to backup previous working image, so it can restore in case this update fails.

Before Secondary installs an update (2)

4. **Verify that the latest image matches the latest metadata.**
    a. Check that the image matches the hash and length for it, obtained from the validated metadata.
    b. If all checks pass, overwrite the previous with the latest metadata. If there is additional storage, overwrite the previous with the latest image.
    c. Otherwise, if some check failed, and there is no additional storage, then restore the previous image from the backup on the primary.
5. **Send the next version report to the primary.**
    a. Include the next token for the time server.
    b. Include the ECU version manifest, which contains: (1) the ECU identifier, (2) the previous and current times, (3) any security attack detected during an update, and (4) metadata about what is currently installed.

Demo!

youtube.com/watch?v=Iz1l7lK_y2c

(or google Uptane Demonstration youtube)