Redesigning PKI
To Solve Revocation, Expiration, & Rotation Problems

Brian Knopf @DoYouQA
WHO AM I

- Sr Director of Security Research & IoT Architect @Neustar @DoYouQA
- 20+ years in IT, QA, Dev & Security
- Home is IoT research lab with 130 devices

Previously
- CEO at BRK Security
- Principal Security Advisor at Wink
- Director of Application Security at Belkin & Linksys
- Principal Test Architect, Office of the CTO at Rapid7
- Director of QA at MySpace
AGENDA

1. Introduction
2. What happened?
3. 140 threat models and the breakthrough
4. Key Terms
5. Components of TDI
6. PKI vs TDI
7. The Workflow
8. The Demo
9. The Code
Introduction
WHAT IS TDI?
What Happened
WHAT HAPPENED
security risks are rising with the exponential growth of connected devices. The company alone has noted a 3,198% increase in attackers prowling for vulnerabilities in IoT devices over the past three years.
**Leading Certificate Authorities and Microsoft Introduce New Standards to Protect Consumers Online**

- **Stronger protection for private keys:** The best practice will be to use a FIPS 140-2 Level 2 HSM or equivalent. Studies show that code signing attacks are split evenly between issuing to bad publishers and issuing to good publishers that unknowingly allow their keys to be compromised. That enables an attacker to sign malware stating it was published by a legitimate company. Therefore, companies must either store keys in hardware they keep on premise hardware, or in a new secure cloud-based code signing cloud-based service.

- **Certificate revocation:** Most likely, a revocation will be requested by a malware researcher or an application software supplier like Microsoft, if they discover users of their software may be installing suspect code or malware. After a CA receives request, it must either revoke the certificate within two days, or alert the requestor that it has launched an investigation.

- **Improved code signatures time-stamping:** CAs must now provide a time-stamping authority (TSA) and specifies the requirements for the TSA and the time-stamping certificates. Application software suppliers are encouraged to allow code signatures to stay valid for the length of the period of the time-stamp certificate. The standard allows for 135-month time-stamping certificates.

Microsoft will require CAs that issue code signing certificates for Windows platforms must adhere to these guidelines beginning on February 1, 2017.
Over Half a Million Belkin WeMo Users

Seattle, US — February 18, 2014 — IOActive, Inc., the leading global provider of specialist information security services, announced today that it has uncovered multiple vulnerabilities in Belkin WeMo Home Automation devices that could affect over half a million[1] users. Belkin’s WeMo uses Wi-Fi control home electronics anywhere in the world directly.

Mike Davis, IOActive’s principal research scientist, used a combination of tools from the WeMo product set that gives attackers the ability to gain administrator access to devices anywhere in the world.

The Vulnerabilities

The Belkin WeMo firmware images that are used to update the devices are signed with public key encryption to protect against unauthorised modifications. However, the signing key and password are leaked on the firmware that is already installed on the devices. This allows attackers to use the same signing key and password to sign their own malicious firmware and bypass security checks during the firmware update process.
140 Threat Models
and the Breakthrough
140 THREAT MODELS AND THE BREAKTHROUGH
• Crypto is hard

• Developers, like everyone else, make mistakes

• Keys expiring on IoT devices will totally ruin your Saturday
REQUIREMENTS FOR PKI REPLACEMENT

• NOC or SOC should be in control, not users/site managers
• Do NOT rely on the router & firewall as your security model
• Trust nothing unless proven otherwise… constantly
• Servers should not share keys for signing
• Revocation should be instant
• Key rotation should be easy & fast
• Keys should never expire unexpectedly
• Plan for complete failure
Key Terms
FLEET

A fleet defines the scope of the deployment. Typically, a Fleet is comprised of devices with a certain SKU and those devices’ supporting services. The public key of the fleet is recognized by all elements of the system and represents a Fleet’s base authority and identity.

CO-SIGNING SERVICE

Messages receive a second signature from the TDI Co-Signing service to strengthen the integrity of the message and its authenticity. The TDI Co-Signing service retains its key pair as well as the public keys for the Fleet Server and Devices.

FLEET SERVER

The private fleet key is stored in an HSM and paired with a server(s) associated with that Fleet of devices. This server acts as a verification point of messages and can sign any messages directed to devices on behalf of the fleet.

DEVICES

Devices are assigned to a fleet. They have the ability to create messages and sign with their private key. They also contain the public key of the fleet and of the co-signer to verify any messages being sent.
FLEET PRIVATE key
Fleet PUBLIC key

FLEET SERVER
Server PRIVATE key
Server PUBLIC key
Fleet PRIVATE key
Device PUBLIC key
Co-Signer PUBLIC key

CO-SIGNING SERVICE
Co-signer PRIVATE key
Co-Signer PUBLIC key
Fleet Server PUBLIC key
Device PUBLIC key

DEVICE
Device PRIVATE key
Device PUBLIC key
Fleet PUBLIC key
Co-Signer PUBLIC key
Reset PUBLIC keys
KEY CONCEPTS

1. Devices and servers each have a unique key pair identity. We recommend generating the key based on a hardware root of trust (Arm Trust Zone or Intel TPM).

2. Devices and servers are assigned to a fleet.

3. Fleet servers can verify trusted devices in the fleet, sign on behalf of the Fleet Signing Key, and request co-signature from the co-signing server using its Fleet Server Key.

Fleet Name: My Parking Lot

Devices send messages signed with their private keys which is validated by fleet server & co-signer.
Components of TDI
REVIEW OF THE COMPONENTS OF TDI

**TDI Co-Signing Engine**
Hosts co-signing service, verifies devices and provides an aggregated view of devices, users & verification gateways

**Client App**
Regular application sending and receiving messages from devices

**Fleet Server**
Hosts Fleet signing service, manages & verifies devices assigned to this identity gateway

**Gateway or Device Firmware/App**

- **Identity Agent**
  Signs messages & verifies that messages are ok
<table>
<thead>
<tr>
<th>TECHNOLOGY</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cryptography</strong></td>
<td>EC NSA Suite B, NIST p256 curve</td>
</tr>
<tr>
<td><strong>Encryption</strong></td>
<td>AES 256</td>
</tr>
<tr>
<td><strong>APIs</strong></td>
<td>RESTful</td>
</tr>
<tr>
<td><strong>SDKs</strong></td>
<td>Python, C, Node, Java</td>
</tr>
<tr>
<td><strong>Communication Protocols</strong></td>
<td>Agnostic</td>
</tr>
<tr>
<td><strong>Device Requirements</strong></td>
<td>Agnostic, &lt;100KB + SSL library, 32-bit CPU, 32K RAM</td>
</tr>
<tr>
<td><strong>Deployment</strong></td>
<td>CLI and Reference Implementations</td>
</tr>
<tr>
<td><strong>Hosting</strong></td>
<td>Cloud (AWS), Private Cloud, On-Premise</td>
</tr>
<tr>
<td><strong>Management</strong></td>
<td>API hooks for NOC/SOC, CLI, Admin Portal</td>
</tr>
</tbody>
</table>
PKI vs TDI
In a typical PKI deployment an identity asserts itself to a recipient.

The recipient looks up the identity in CRL or via OCSP and then it’s own cert list to validate.

Burden on recipient!

In a TDI deployment the Identity service validates identities and returns the validation to your application.

Only the validation is forwarded to the recipient.

Recipient does not need to do additional CRL/OCSP lookup.
PKI WITH CRL & SHARED SERVER KEYS

1. Single server key compromise causes all servers to be compromised.
2. Compromised server keys are added to CRL.
3. Devices have to lookup CRL to identify expired keys.
4. Compromised keys need to be removed from every server and device that could communicate with them.
PKI WITH CRL & INDIVIDUAL SERVER KEYS

1. Only server(s) compromised need to be revoked
2. Compromised server keys are added to CRL
3. Devices have to lookup CRL to identify expired keys
4. Compromised keys need to be removed from every server and device that could communicate with them
5. Every device and server need an update with the public key every time a server is added or removed
PKI WITH OCSP STAPLING & INDIVIDUAL SERVER KEYS

1. Only server(s) compromised need to be revoked
2. Compromised server keys no longer validated with OCSP
3. Devices don’t have to lookup CRL, but still have public keys on them
4. Compromised keys still need to be removed from every server and device that could communicate with them
5. Every device and server still need an update with the public key every time a server is added or removed
SO WHAT DOES THIS GIVE US?

**TDI**

- Servers & devices can be instantly revoked.

**PKI**

- Devices must check to see if server is revoked. Burden on end device for lookup.

**TDI**

- Devices only need to store and validate 2 keys to receive messages *(co-signer and fleet public keys)*

**PKI**

- Devices must store a key(s) for each of the entities it is communicating with

**TDI**

- Servers can be quickly switched out without rekey of all devices because devices only respect the co-signer and fleet-signer keys

**PKI**

- Potentially 1,000’s of devices need to be re-keyed with the new server key
The Workflow
Owner creates a TDI admin account and creates a new Fleet in the TDI service (using public key from HSM)

Owner retrieves Fleet public key from HSM

TDI creates Fleet repo and TDI Fleet key pair
Owner sends to new Server
- Fleet public key
- TDI public key
Server has access to HSM for signing
(its own sig and Fleet sig)

Owner retrieves new Server public key from HSM and sends to TDI Fleet repo
- Server public key
PROVISIONING DEVICES

1. Owner sends to device
   - TDI public key
   - Fleet public key
   - 3 reset public keys
   If Device keys generated by Owner, also sends
   - Device key pair
   Else if Device has SE
   - SE public key retrieved as Device public key

2. Owner sends to Fleet repo and oneID Fleet repo
   - Device public key
HOW IT WORKS

Server to Device Messaging

1. Server generates a request, signs it with Server key, and sends to TDI cloud for co-signature of payload

2. TDI Service validates Server signature...

3. ...and returns the request signed with TDI cloud Fleet private key

4. Server validates TDI signature and applies Fleet signature
   - Now 2 Signatures on the message
   - Server signature still on request

5. Server forwards the request, signed by Fleet and TDI service

6. Device validates Fleet and TDI signatures before accepting request

TDI service provides a central repository for the public keys of all entities in a fleet. The TDI service is called via RESTful APIs.
Device to Server Messaging

1. Device generates a request and signs it with Device private key
2. Server validates Device Signature with Fleet repo and forwards the request to co-signer service
3. TDI service validates the device signature...
4. ...and returns the request signed with TDI cloud Fleet private key
5. Server validates co-signer signature before accepting the message
SIMPLIFIED MESSAGE FLOW: DEVICE SENDING A MESSAGE

1. The Device generates a message and signs the message with its private Device Signing Key.

2. App ingests the message and sends to the fleet server for validation and co-signatures.

3. Fleet Server verifies the signature against the public Device Verification Key.

4. Co-Signer verifies that neither the fleet server nor the device have been revoked.

5. Co-Signer signs the message and sends it back to the fleet server.

6. Fleet Server verifies the co-signature key & sends to destination app.

7. App does whatever it wants with the message.
SIMPLIFIED MESSAGE FLOW: APP SENDING A MESSAGE TO A DEVICE

1. Message is generated by the app, signed with it’s App/Device key, and sent to the fleet server – e.g. firmware

2. Fleet server verifies the app and sends to co-signature service

3. Co-Signer verifies that neither the fleet server nor the app server have been revoked & signs the message

4. Sends the message back to the fleet server

5. Verifies the co-signature key and returns the dual signed message to the app

6. App sends message to device

7. Device verifies fleet signature & co-signer signature and carries out operation
TDI enables rapid breach recovery with minimal downtime

1. Admin revokes compromised Server(s) in their TDI fleet via CLI / dashboard or API
2. Admin provisions new Servers into the Fleet
3. On the next request the old servers are no longer valid and new servers are immediately validated by the TDI service
4. Recipients DO NOT require updates for servers being revoked or added
In the rare occurrence of needing to re-key, **3 reset public keys are stored with provisioned identities to enable backup MFA.** This means that both the fleet and co-signer private keys can be compromised and recovery can still be done remotely.
ROBUST AUTHENTICATION

AUTOMATED MFA
servers, devices, people, and applications automatically establish trust with one another using n-factor authentication.

BIDIRECTIONAL AND MUTUAL requests are authenticated whether sent upstream or down, providing data provenance. TDI also supports mutual auth if required for your application.

FLEXIBLE MANAGEMENT

GRANULAR IDENTITY MANAGEMENT
every server, device, user, and service is provisioned with a unique key, so each and every identity can be managed in real time.

REAL-TIME PROVISIONING / REVOCATION
add identities in real-time without sharing public keys with every recipient. revoke in real time as well, without distributing crls or requiring ocsp calls by recipients.
THE GRAVY
Validate the path a message takes to ensure that it originated from a proper location in your network.

Devices, gateways, and routers each sign with their own keys.

Server validates the route, each individual key, and the order of keys.
Future Framework Enhancements
Server sends to site trends to Neustar & TDI for asset verification, anomaly detection, & analysis.

Neustar validates sites & identifies large changes in traffic patterns. TDI validates signature and behavior. TDI servers can be replicated to withstand DDoS.

Gateway validates asset legitimacy, policy legitimacy, and passes site information upstream.

Admin uses CLI/dashboard or API to monitor activity.

Update black/white lists at device or gateway, authorized by TDI.

TDI pushes whitelist & blacklist updates down to gateway & device for mitigation.
Hierarchal fleet signer & local co-signer to allow for offline TDI messages you can use the cloud co-signer to grant permission for a certain period of time

Cache messages for a period of time, so messages are pre authorized for a period of time

Device to device set a time to live for on ECDHE derived key message exchange between devices, gateways & users. This can be set up in Fleet server of cloud administration.
The Demo
The Code
THE CODE

OPEN SOURCE CODE
https://github.com/Neustar-TDI

DOCUMENTATION

MORE INFO ON TDI

QUESTIONS & SUPPORT
earlyaccess.iot @ team.neustar
ACKNOWLEDGEMENTS

- Casey Newton
- Steve Kirsch
- Neustar IoT Team
- Callie Holderman for editing the demo
- Countless security researchers who reviewed this
- My wife and kids for their endless patience & support
Redesigning PKI To Solve Revocation, Expiration, & Rotation Problems

Brian Knopf @DoYouQA

eustar.

- Do NOT rely on the router & firewall as your security model
- Trust nothing unless proven otherwise… constantly
- Enable your SOC or NOC to control the security rather than users or site managers
Thank You