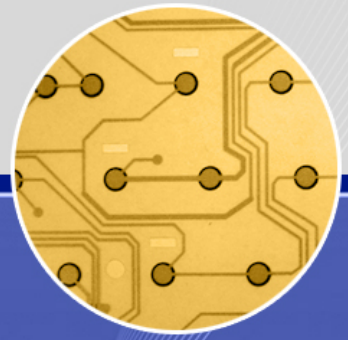


BENINGO EMBEDDED GROUP

Secure Bootloader Design Techniques for MCU's

atollic

Embedded RELATED.com

The Lecturer

Jacob Beningo

Principal Consultant

Social Media / Contact

- E : jacob@beningo.com
- T : 810-844-1522
- : Jacob_Beningo
- : Beningo Engineering
- : JacobBeningo
- EDN : Embedded Basics
- ARM Connected Community

Newsletters

- Embedded Bytes

http://bit.ly/1BAHYXm

Consulting

- Secure Bootloaders
- Code Reviews
- Architecture Design
- Real-time Software
- Expert Firmware Analysis
- Microcontroller Systems

Embedded Training

- RTOS Workshop
- Bootloader Design
- Debugging Techniques
- Security Fundamentals
- Micro Python

www.beningo.com

Session Overview

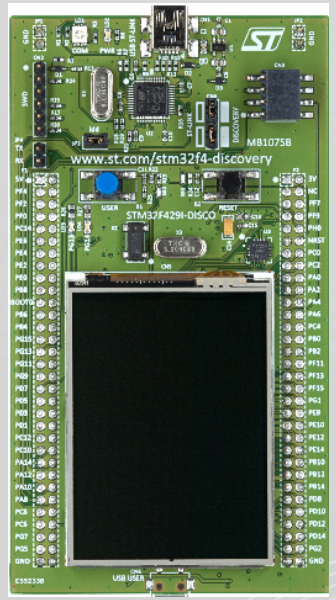
Objective:

- Explore bootloader design and the techniques necessary to secure them.

Topics:

- Implementation models
- Bootloader fundamentals review
- Secure bootloader stages
- Setting a security strategy
- Key concepts for security, robustness and speed
- Best practices for secure bootloader design

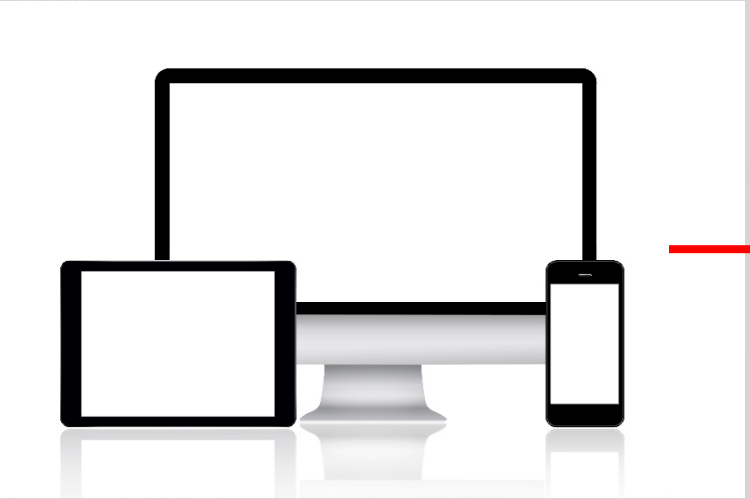
Hands-on Example Materials



Visit www.atollic.com to download the materials

What is a bootloader?

Environment



MCU

Where do bootloaders come from?

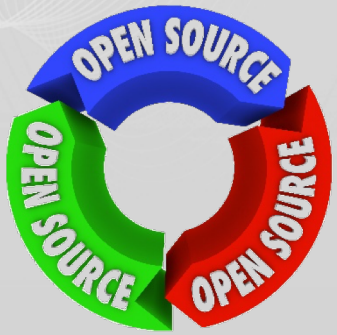
1

Silicon Vendors

2

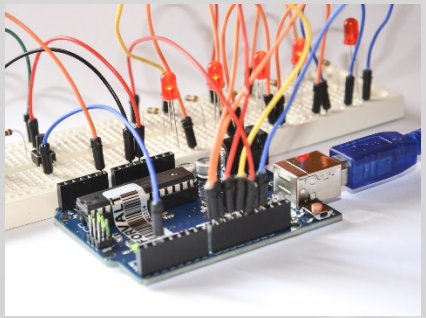
3rd Party

3



Open Source

4

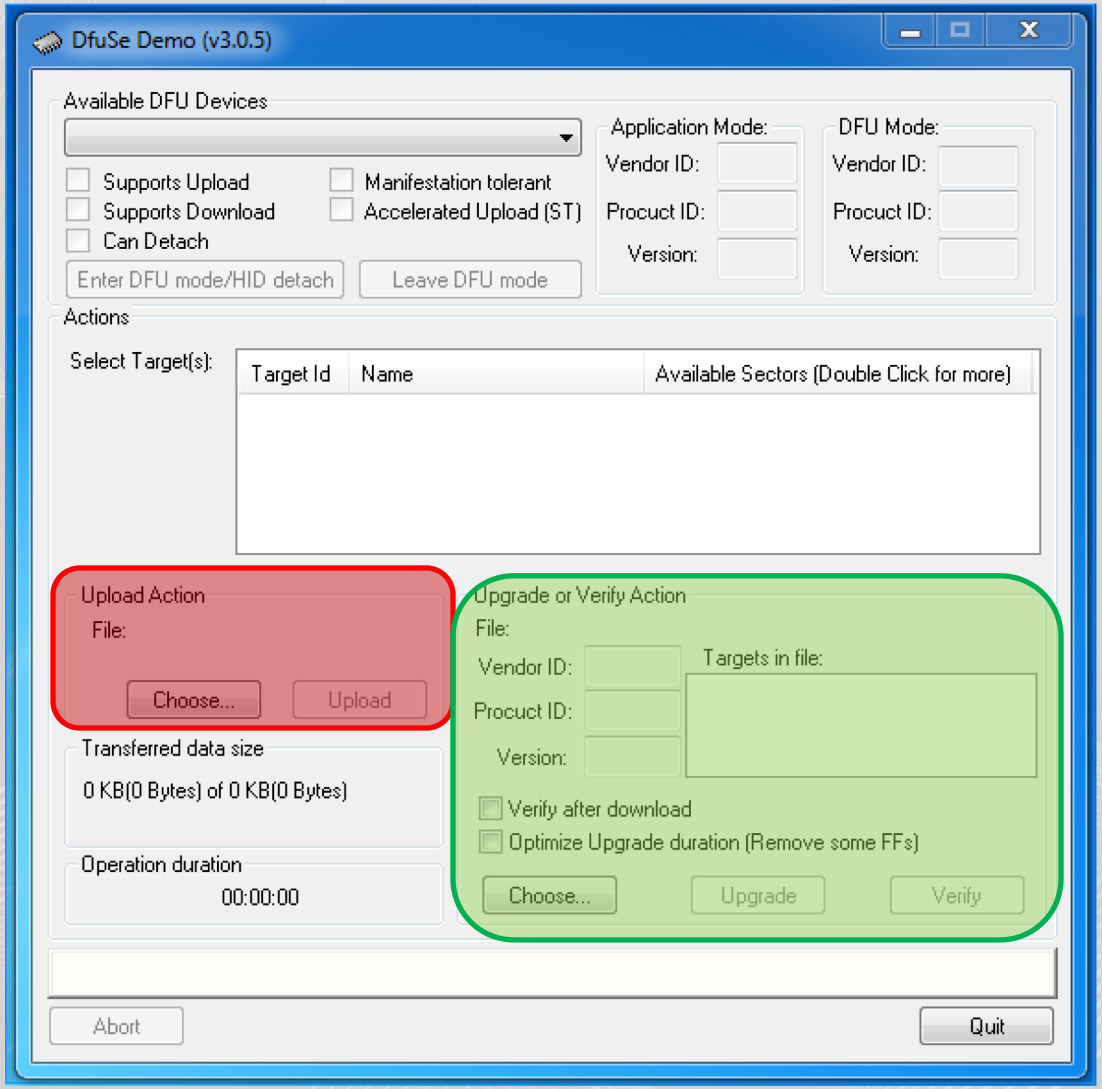


Roll your own

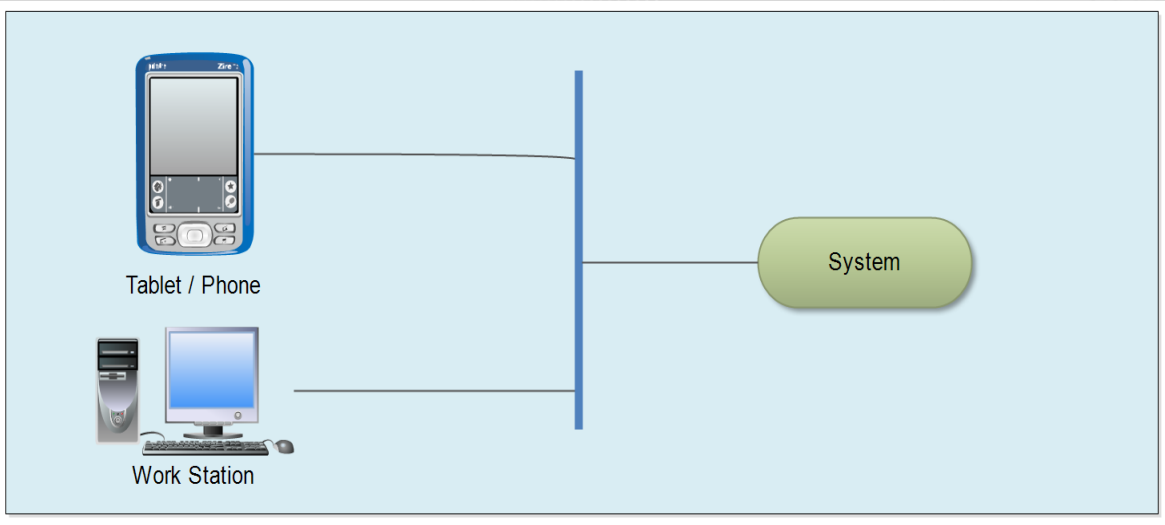
DfuSe Bootloader Utility

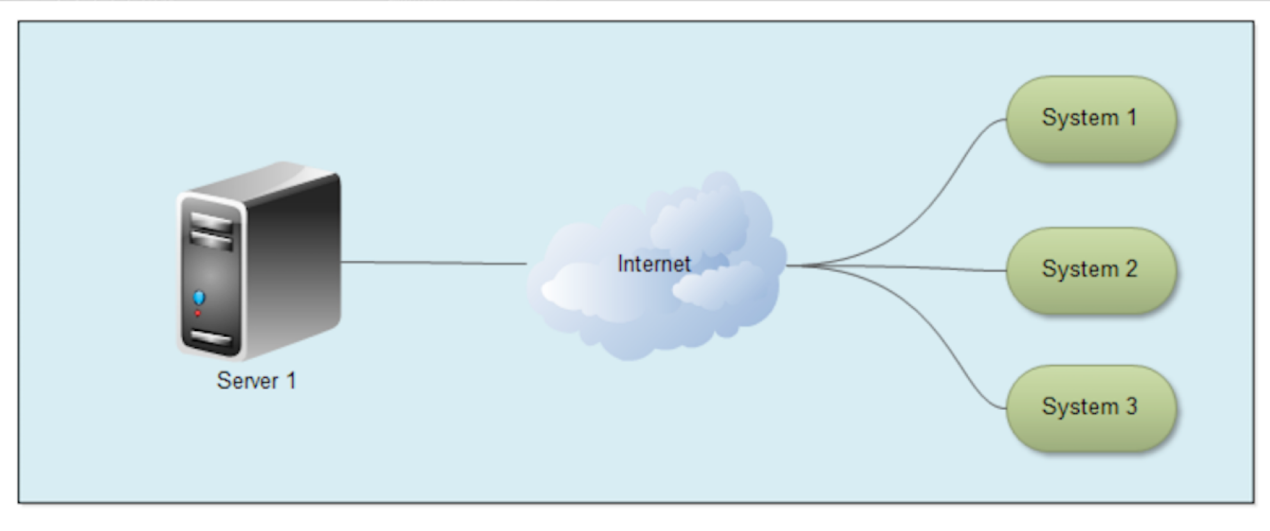
BENINGO EMBEDDED GROUP

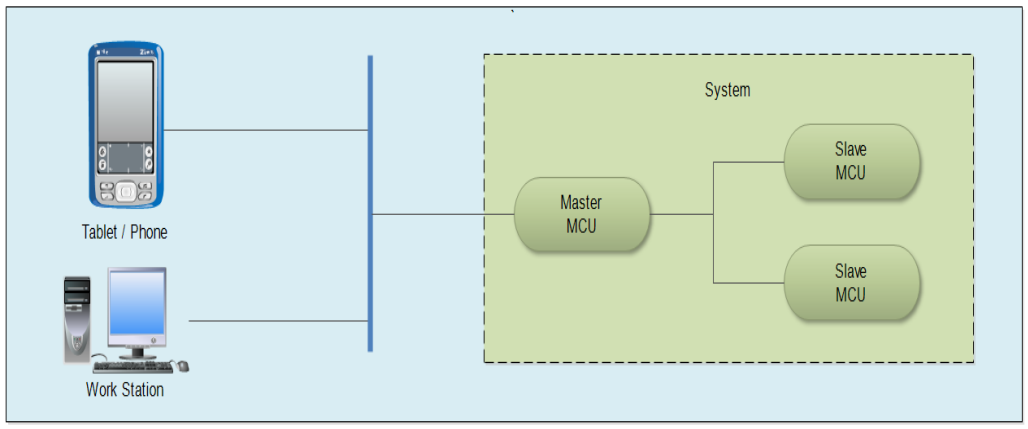
STM32 ARM Cortex-M

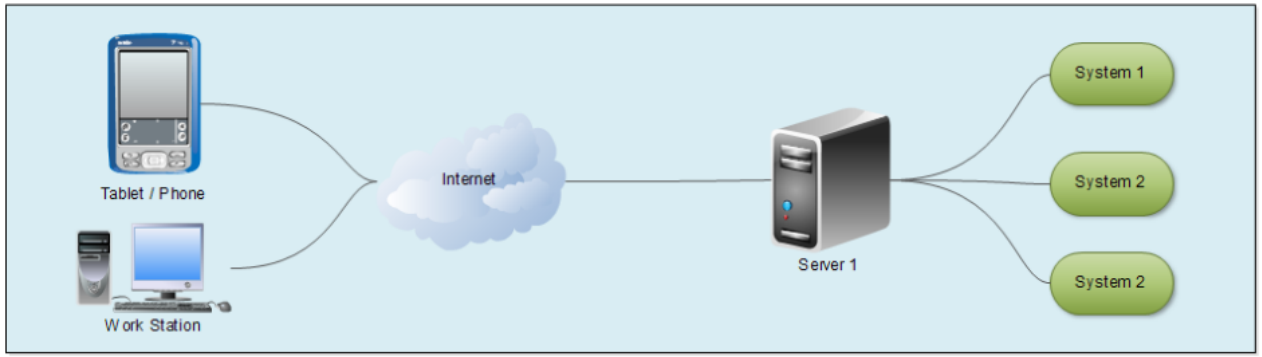


Basic Bootloader Models







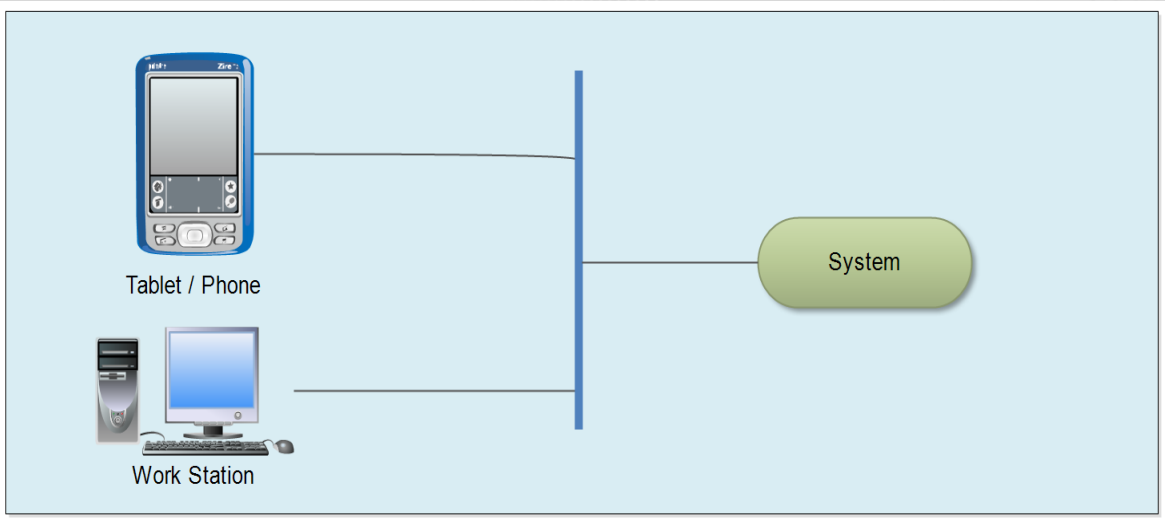


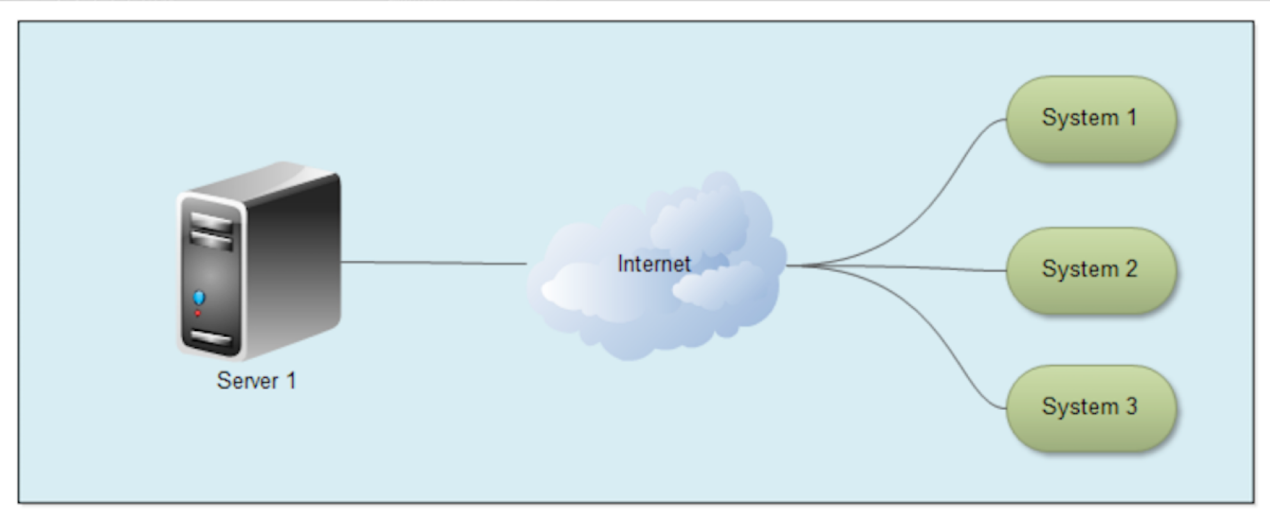
DEMO Project Setup

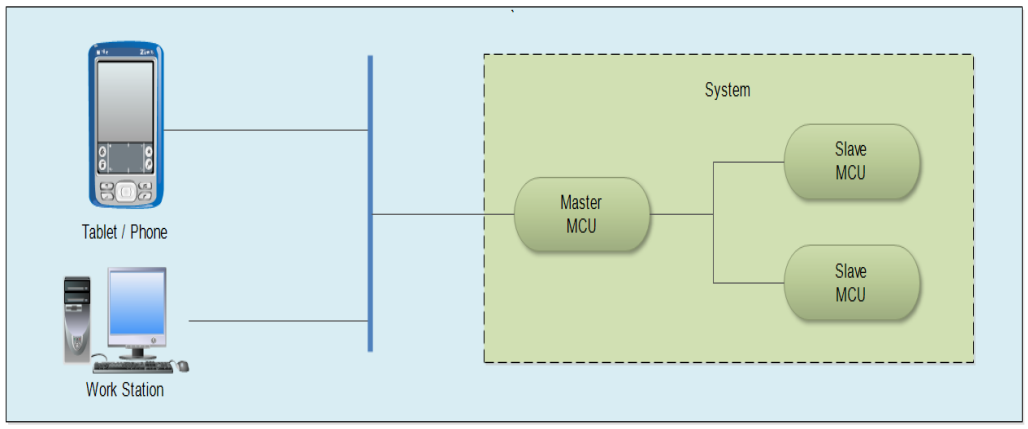
BENINGO EMBEDDED GROUP

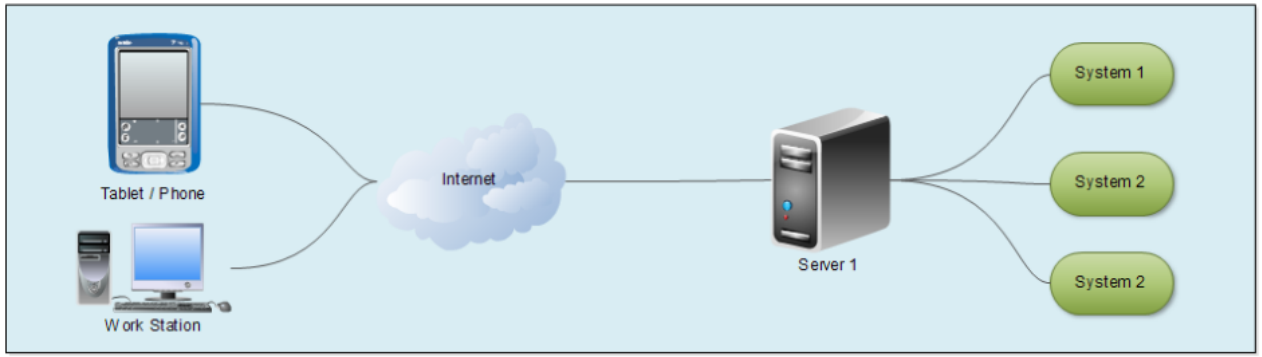
Bootloader Demo

Basic Bootloader Models

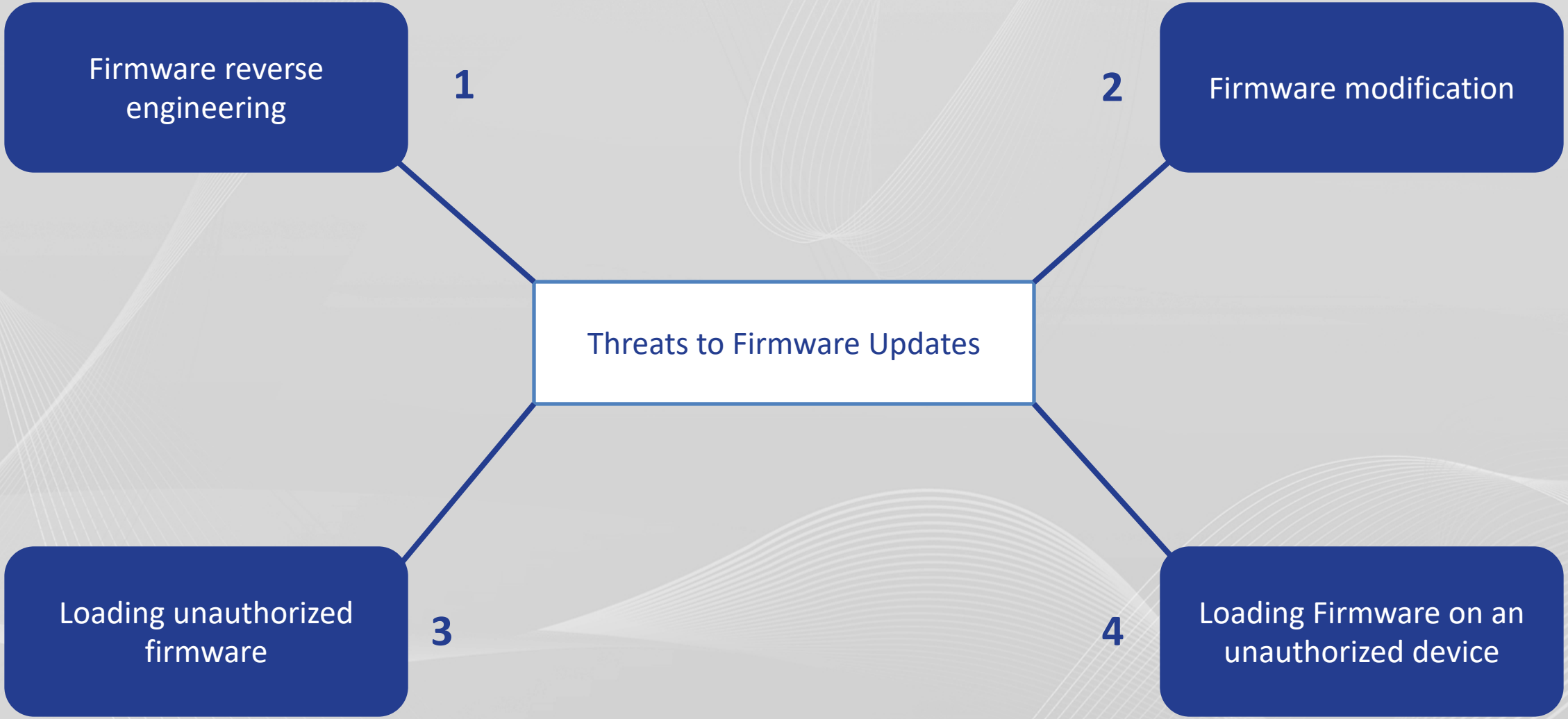






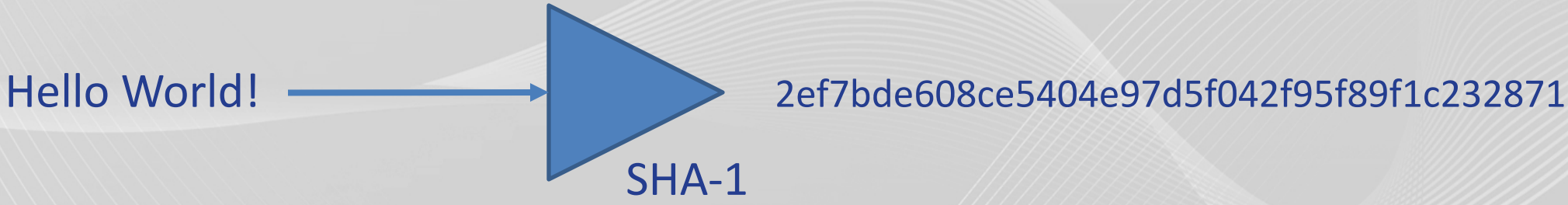


The need for secure updates



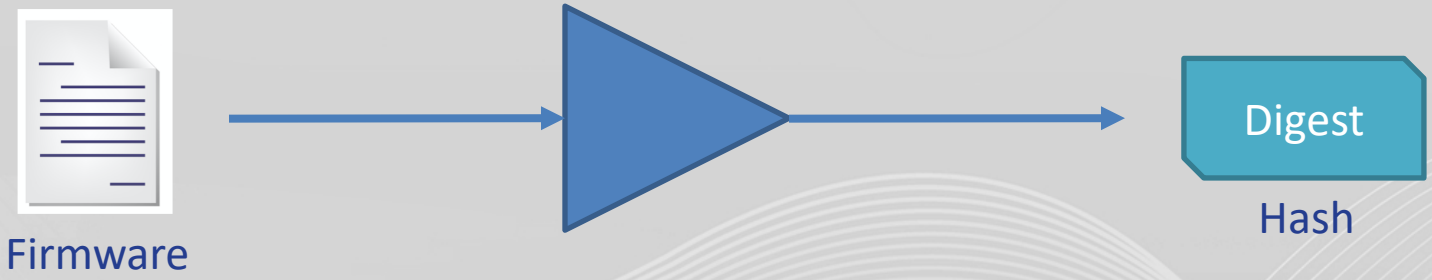
Hash Algorithms

- Hashing
  - An algorithm that produces a digital “fingerprint” of the data that it processes
  - Is a one-way function (not reversible)
  - Takes a string of any length and generates a fixed-size output (digest)
- Example hash functions
  - MD5 (120-bit)
  - SHA-1 (160-bit)
  - SHA-3 (224-bit, 256-bit, 512-bit)

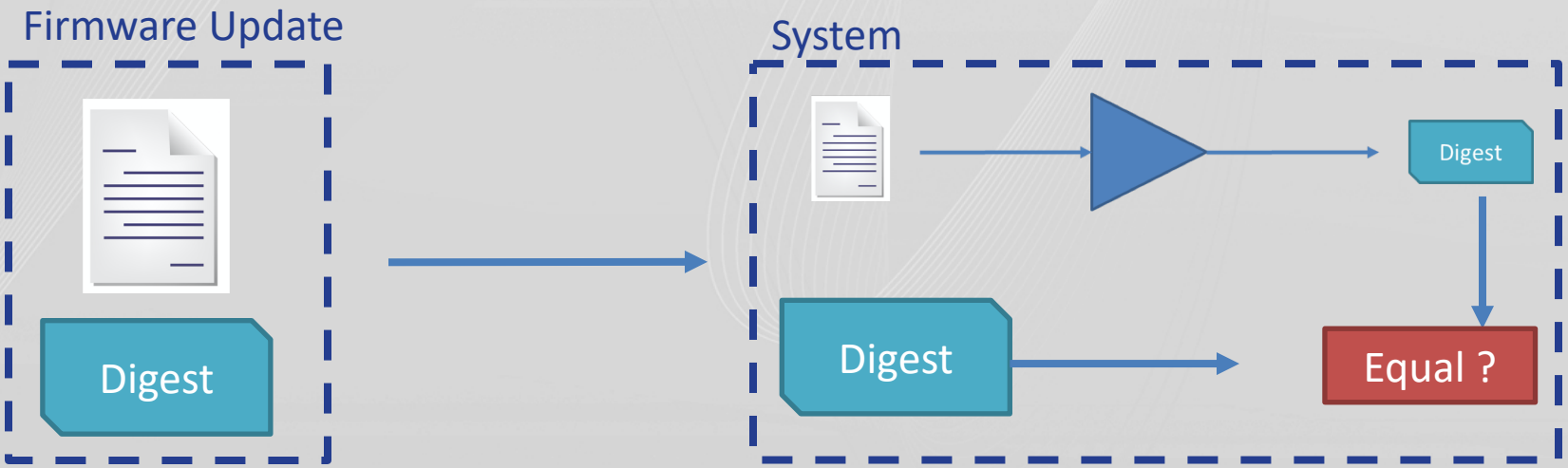


Hash Algorithms

- Hashes can be used to verify the firmware
  - Does the received hash match the received firmware?
  - On startup, does the application match what is expected?



Hash Algorithms



- Potential Issues
  - Hacker can modify software and recalculate the hash
  - Additional overhead to calculate
  - May require more code space

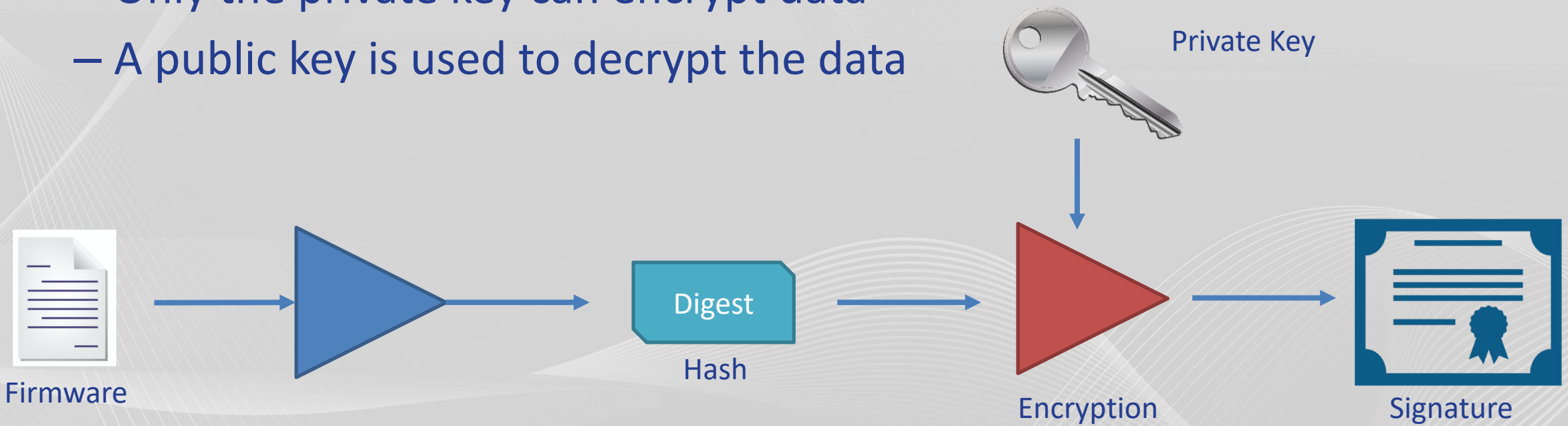
Hash Algorithms

- Speed and Size Comparisons

Algorithm	Block Length	Security Status	Compute Speed
MD5	128 bits	Broken	Fast
SHA-1	160 bits	Broken	Slow
HAVAL	128 to 256 bits	Broken	Fast
SHA3-256	256 bits	Secure	Slow
Tiger	192 bits	Secure	Fast
WHIRLPOOL	512 bits	Secure	Super Slow
RIPEMD-160	160 bits	Secure	Slow

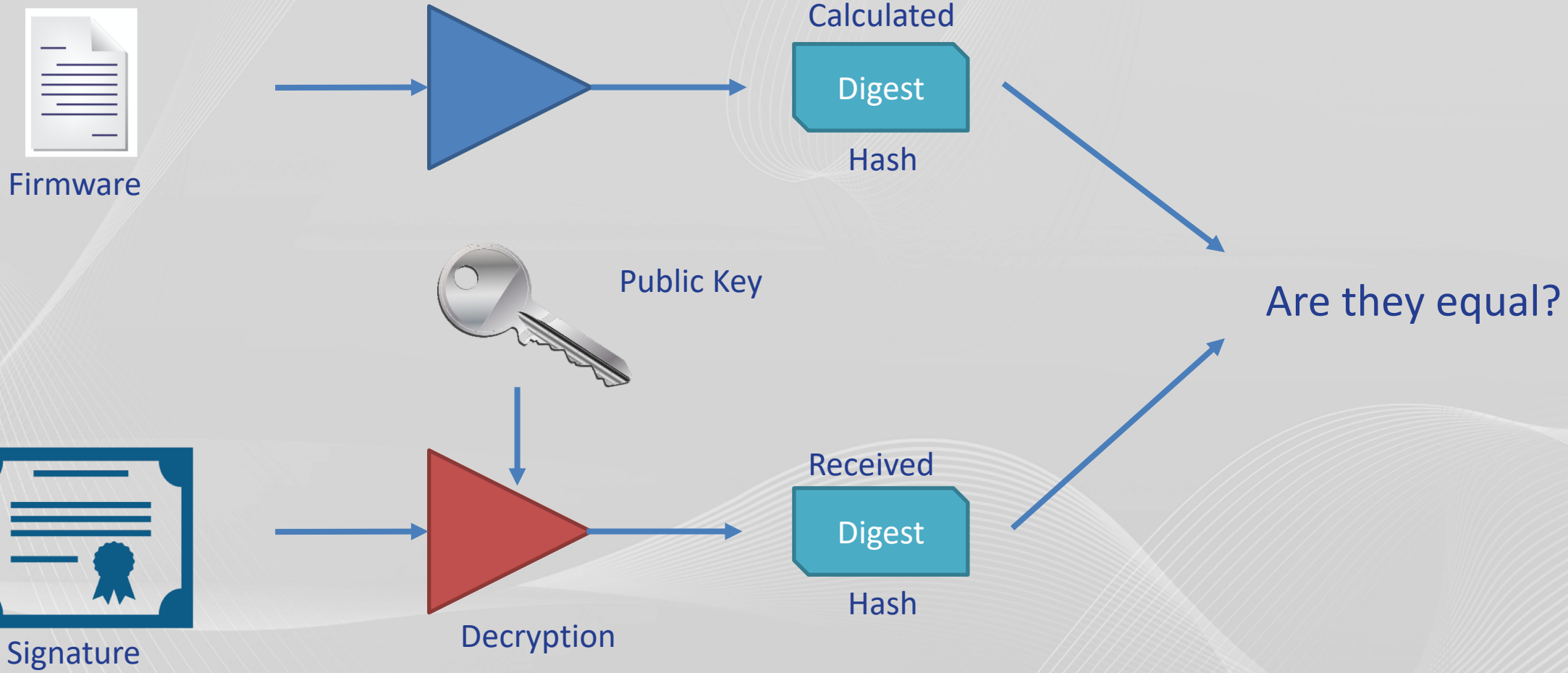
Digital Signatures

- Digital Signature (Digital Signing)
  - Encrypting the firmware digest using asymmetric encryption
    - Ex: RSA
  - Only the private key can encrypt data
  - A public key is used to decrypt the data



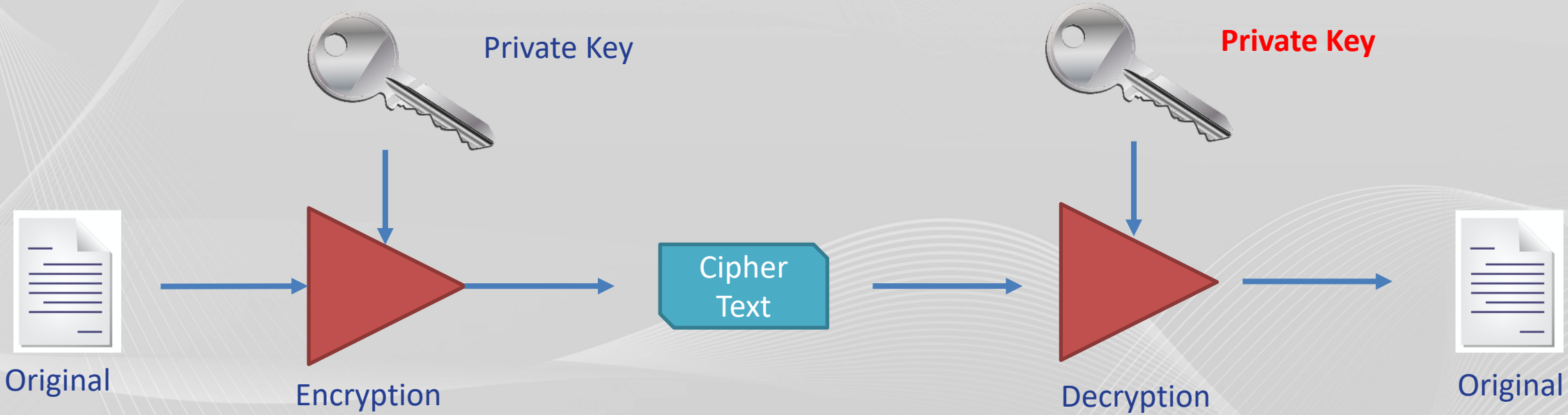
Digital Signatures

- Authentication



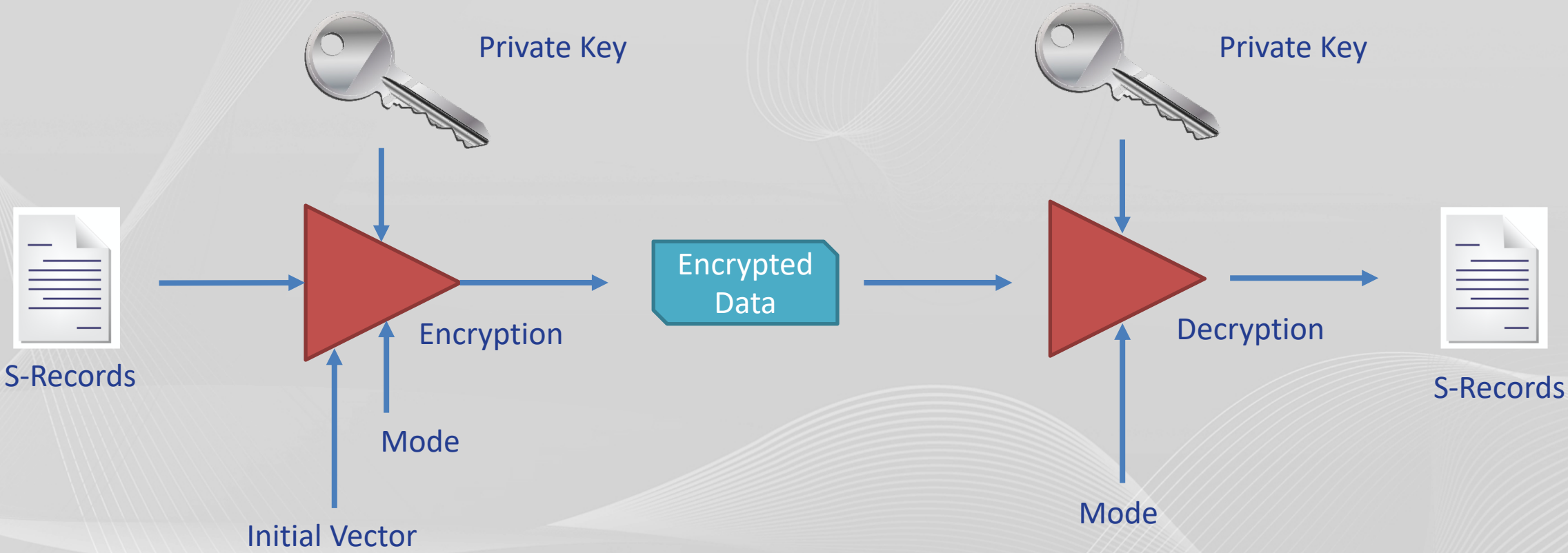
Cryptography Overview

- Secret Key Cryptography
  - Also known as Symmetric Key
  - Single private key exchanged between device and manufacturer



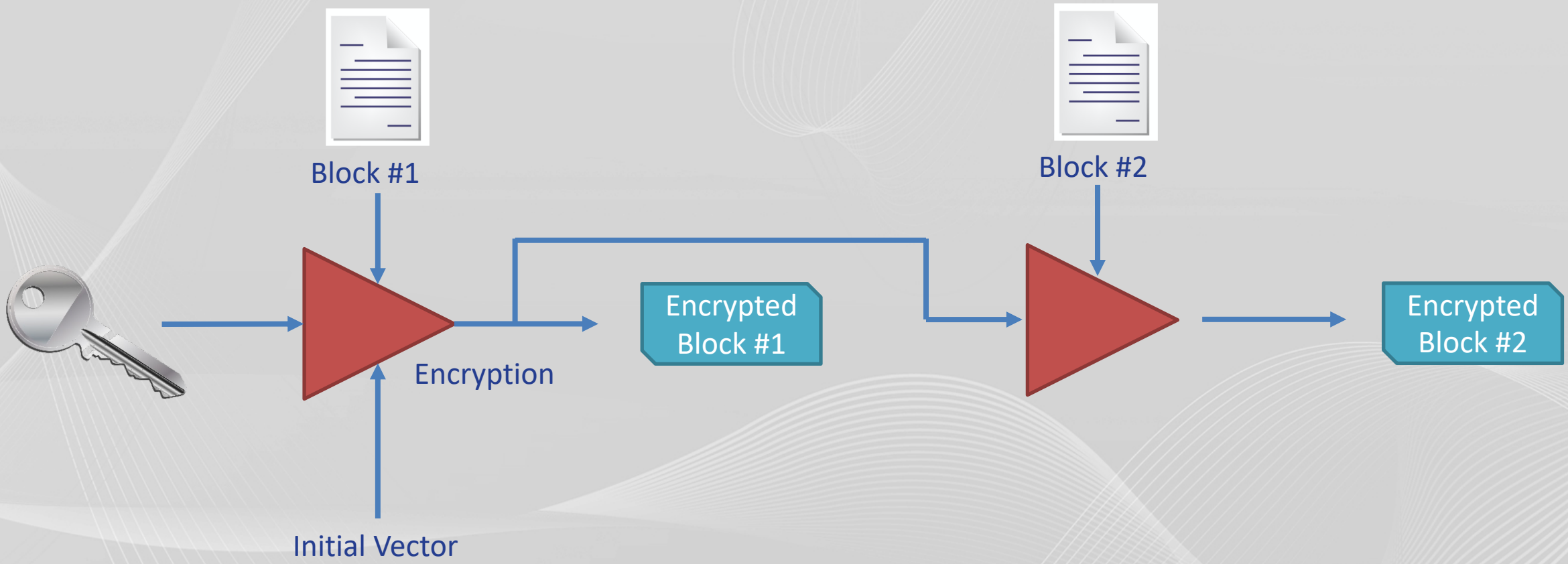
Cryptography Overview

- Example AES Implementation (Cipher block chaining [CBC])



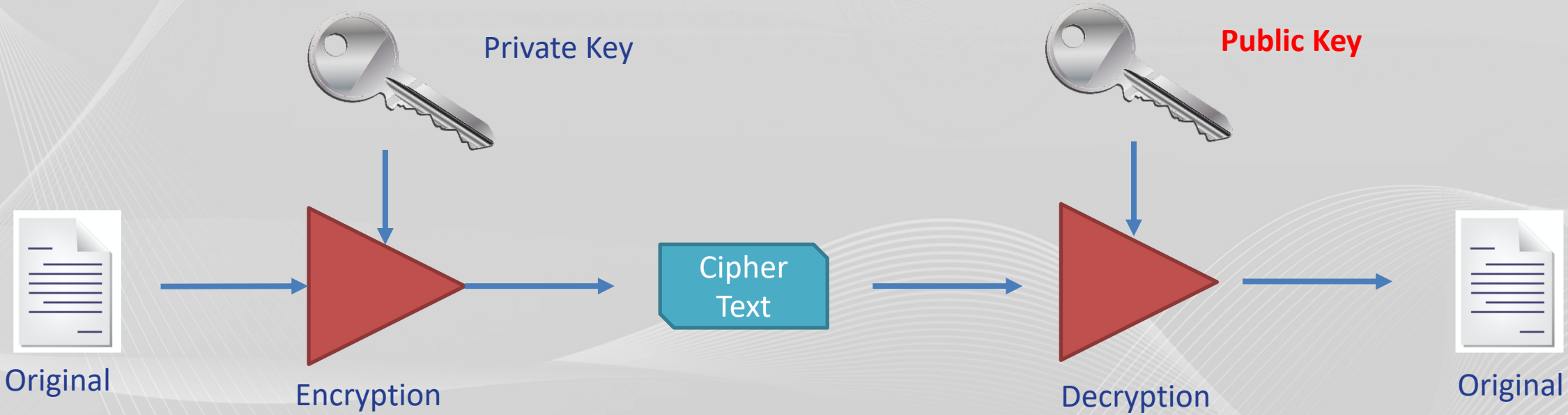
Cryptography Overview

- Example AES Implementation (Cipher block chaining [CBC])



Cryptography Overview

- Public Key Cryptography
  - Also known as Asymmetric Key
  - Private key is used to encrypt
  - Public key is used to decrypt

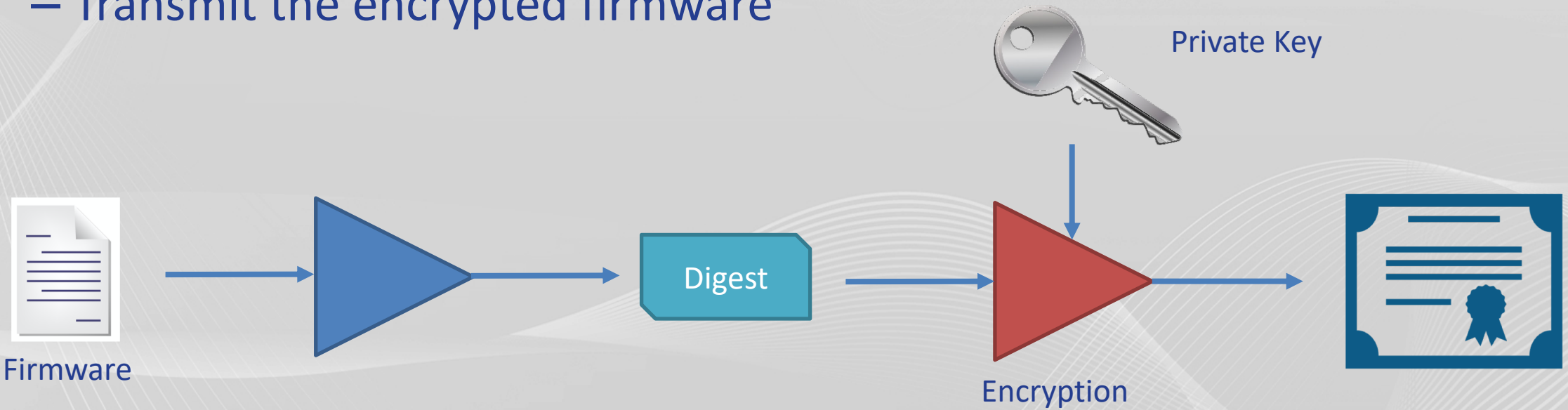


Developing a Secure Bootloader

- Several different methods
  - Download entire encrypted image before processing
  - Download single encrypted records and partial program
  - Transmitting an unencrypted application over a secure protocol such as TLS
  - Combination of the above
- Considerations
  - Speed
  - Code size
  - Security level requirements

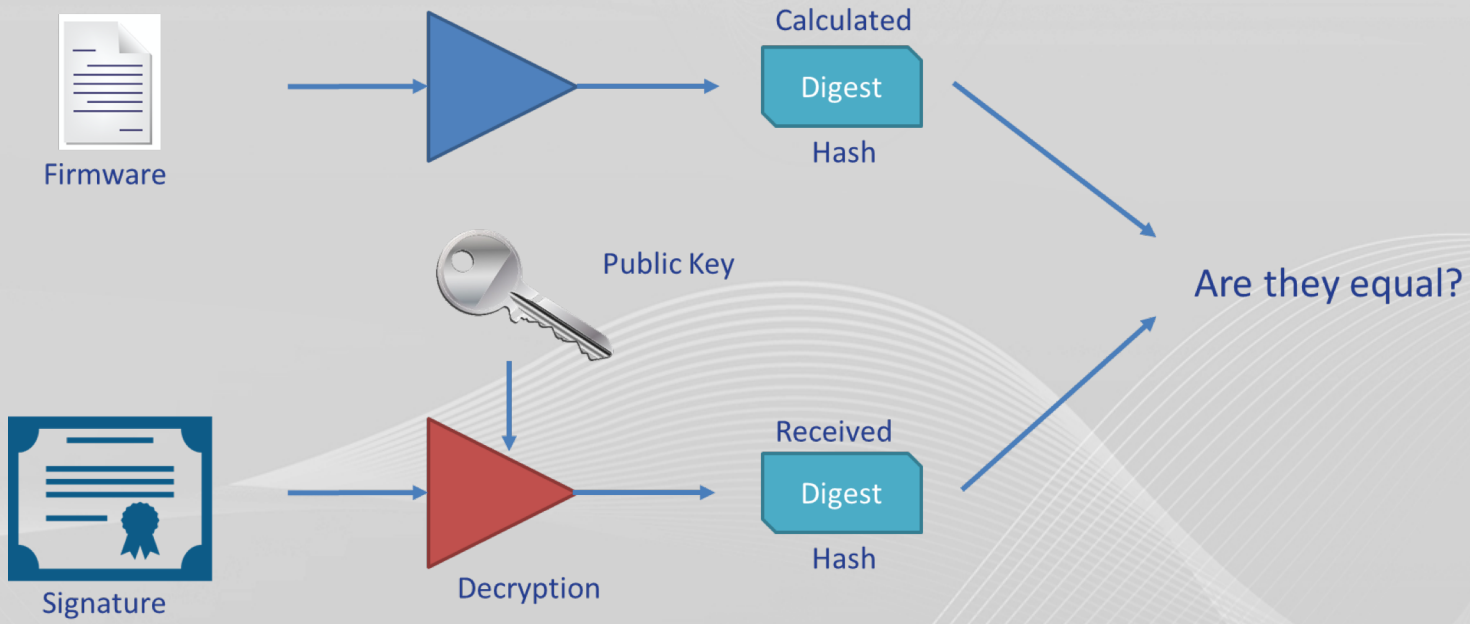
Developing a Secure Bootloader

- Manufacturer Side
  - Generate a signature for firmware authentication and verification
  - Encrypt the firmware
  - Transmit the firmware signature
  - Transmit the encrypted firmware



Developing a Secure Bootloader

- Device Side
  - Store encrypted image and signature to temporary memory
  - Decrypt the signature and authenticate
  - Decrypt the image and write to flash



Securely Boot the System

- Generate a “Root of Trust”
  - Initialize stage 1 trusted software (bootloader)
  - Authenticate resident application signature
  - Authenticate device for application
  - Load application

Secure Bootloader Best Practices

- Select a microcontroller that has
  - a hardware cryptographic engine
  - True random number generator (TRNG)
  - Memory Protection Unit
- Lock the flash security bits to protect the bootloader and application
- Securely store private keys
- Clearly identify up-front the level of security that is necessary for the bootloader
- Use signatures to authenticate the firmware source
- Minimize complexity
- Use AES to encrypt your firmware

Going Further

- Download beningo.com resources
  - C Doxygen templates
  - RTOS Best Practice Guide
  - Bootloader White Paper
  - Bootloader Design Techniques Course
- Atollic.com resources
  - TrueStudio 8.1
  - Bootloader template project
- EmbeddedRelated.com
  - Embedded Question Forum

Questions