S32K344 Secure Boot Implementation Guide

Document Version: 1.0
Date: 2025-05-07
Author: SignalSlinger

Table of Contents

Introduction

This document describes the technical implementation of secure boot on the NXP S32K344 microcontroller using Hardware Security Engine (HSE). The implementation provides a robust security framework that ensures only authenticated firmware is executed on the device.

Architecture Overview

The S32K344 secure boot implementation leverages the following components:

- **Hardware Security Engine (HSE)**: Hardware-based security module
- **ECC P-256 Cryptography**: For digital signatures
- **LED Indicators**: Visual feedback for boot states
- **Boot Configuration Table**: Configuration for secure boot parameters

The secure boot process verifies the authenticity and integrity of the application firmware before execution, preventing unauthorized or tampered code from running.

Implementation Components

Core Components

1. **HSE Configuration Files**
   - `hse_config.h`: Definition of HSE configuration structures
   - `hse_config.c`: Implementation of HSE configuration and boot sequence
2. **Key Management**
   - Public/private key handling for firmware authentication
   - Key generation tools and procedures
   - Signature generation and verification
3. **Application Integration**
   - `main.c`: Application entry point with boot status indicators
   - `HSE` integration code: Interface between application and HSE

Directory Structure

```
C:.
├── .settings                    # IDE settings
├── board                        # Board-specific configurations
├── Debug_FLASH                  # Debug build output
│   ├── board
│   ├── generate
│   ├── hse_config
│   ├── Hse_Files
│   ├── keys
│   ├── Project_Settings
│   └── src
├── generate                     # Generated code
│   ├── include
│   └── src
├── hse_config                   # HSE configuration files
├── Hse_Files                    # HSE-specific files
│   └── dcm_register
├── keys                         # Key storage
├── Project_Settings             # Project configuration
│   ├── Debugger
│   ├── Linker_Files
│   └── Startup_Code
├── Release_FLASH                # Release build output
├── RTD                          # Run-time drivers
│   ├── include
│   └── src
└── src                          # Main source code
```

Key Generation & Management

Key Generation Process

The implementation uses ECC P-256 (NIST curve) for cryptographic operations. Keys are generated using standard OpenSSL commands:

1. **Generate ECC Key Pair**

   ```
   openssl ecparam -name prime256v1 -genkey -noout -out private_key.pem
   ```

2. **Extract Public Key**

   ```
   openssl ec -in private_key.pem -pubout -out public_key.pem
   ```

3. **Convert to Required Formats**

   ```
   openssl ec -in public_key.pem -pubin -outform DER -out public_key.der
   ```

Signature Generation

Application binaries are signed using the following process:

```
openssl dgst -sha256 -sign private_key.pem -out signature.bin application.bin
```

Key Storage

- **Private Key**: Stored securely off-device, used only during firmware signing
- **Public Key**: Embedded in the application as a C array in `public_key.h`
- **Generated Keys**: Stored in the `keys` directory during development

Secure Boot Flow

The secure boot sequence implemented in `hse_config.c` follows these steps:

1. **Initialization**: HSE subsystem is initialized
2. **Key Validation**: Public key is validated
3. **Signature Verification**: Firmware signature is verified against the embedded public key
4. **Access Control**: Memory protection is configured
5. **Boot Decision**: Boot proceeds only if signature verification succeeds

Boot Status Indicators

LED indicators provide visual feedback on boot status:

- **GREEN LED (slow blink)**: Successful secure boot
- **RED LED (rapid blink)**: Boot failure or signature verification error

Memory Configuration

The implementation configures the following memory regions:

- **Code Flash**: Contains application code with verification at boot
- **Data Flash**: Secure storage for sensitive configuration
- **RAM**: Partitioned into secure and non-secure regions

Memory Protection

Memory protection is configured to prevent:

- Execution from data regions
- Unauthorized access to secure regions
- Modification of critical boot configuration

Implementation Details

Signature Verification

Signature verification uses the following approach:

```c
/* Verify firmware signature */
status = HSE_VerifySignature(
    firmware_address,
    firmware_size,
    signature_address,
    signature_size,
    public_key_address
);
```

Boot Configuration Table

The Boot Configuration Table (BCT) contains critical settings:

```
/* Boot Configuration Table */
static const hse_boot_config_t boot_config = {
    .verification_mode = HSE_VERIFY_ALWAYS,
    .recovery_attempts = 3,
    .boot_timeout_ms = 1000,
    /* Additional configuration */
};
```

Error Handling

Error handling includes:

1. Status code evaluation
2. Visual indicator activation
3. Containment of boot failures

References

1. NXP S32K3xx Reference Manual
2. HSE Firmware Reference Guide
3. AUTOSAR 4.7 Specification
4. NIST SP 800-56A (Cryptographic Key Establishment)

Appendix: Key Functions Overview

Function	Description
HSE_SecureBoot_Init()	Initializes secure boot process
HSE_VerifySignature()	Verifies firmware signature
HSE_ConfigureMemoryProtection()	Sets up memory protection
``	

````markdown name=2_Deployment_Guide.md

# S32K344 Secure Boot Deployment Guide

**Document Version:** 1.0
**Date:** 2025-05-07
**Author:** SignalSlinger

## Table of Contents
````

Introduction

This guide provides step-by-step instructions for deploying the secure boot solution on S32K344 microcontrollers. By following this guide, you will be able to securely program devices with authenticated firmware that can only be updated through authorized channels.

Prerequisites

Before beginning deployment, ensure you have:

- NXP S32 Design Studio (v2023.01 or later)
- OpenSSL (v1.1.1 or later)
- S32 Debug Probe or compatible programmer
- S32K344-EVB or custom hardware with S32K344
- Windows PC for key generation and signing

Required Files

Ensure you have the following project files:

- Complete project directory structure
- Binary application image to be secured

Deployment Steps

The secure boot deployment follows these key steps:

1. Generate cryptographic keys
2. Configure secure boot parameters
3. Sign the application binary
4. Program the device
5. Verify secure boot operation

Key Generation and Management

Step 1: Generate Key Pair

1. Open Command Prompt as Administrator
2. Navigate to the project keys directory
3. Use OpenSSL to generate keys:

```
openssl ecparam -name prime256v1 -genkey -noout -out private_key.pem
openssl ec -in private_key.pem -pubout -out public_key.pem
openssl ec -in public_key.pem -pubin -outform DER -out public_key.der
```

4. Verify the following files are created in the `keys` directory:
   - private_key.pem (KEEP SECURE - never share this file)
   - public_key.pem
   - public_key.der

Step 2: Secure Key Storage

1. Store the private key (private_key.pem) in a secure location
2. Implement proper access controls:
   - Use hardware security modules (HSMs) for production environments
   - Restrict access to authorized personnel only
   - Consider key backups with proper security controls

> ***CRITICAL:*** *The private key must be protected. Compromise of this key would allow unauthorized firmware to be accepted by devices.*

Firmware Signing

Step 1: Prepare Application Binary

1. Build your application in S32 Design Studio
2. Locate the output binary file (typically in the Debug_FLASH directory)

Step 2: Sign the Binary

1. Generate a signature for the binary:

   ```
   openssl dgst -sha256 -sign private_key.pem -out application.sig application.bin
   ```

2. Convert the signature to a C header file:

   ```
   # Use appropriate tool to convert binary to C header
   # Example placeholder - implement with your specific tool
   bin2c application.sig signature.h g_signature "Application signature"
   ```

Step 3: Convert Public Key to C Header

1. Convert the public key to a C header:

   ```
   # Use appropriate tool to convert binary to C header
   # Example placeholder - implement with your specific tool
   bin2c public_key.der public_key.h g_publicKey "Public ECC key"
   ```

Step 4: Rebuild with Signature

1. Place the generated header files in your project:
   - Copy public_key.h to the hse_config folder
   - Copy signature.h to the hse_config folder
2. Rebuild the project to incorporate the signature
3. Verify the build completes successfully

Device Programming

Step 1: Connect Programming Hardware

1. Connect S32 Debug Probe to the target board
2. Ensure proper power supply to the board
3. Connect the debug probe to your PC via USB

Step 2: Program the Device

1. Launch S32 Design Studio
2. Select "Flash" perspective
3. Configure the flash tool:
   - Select S32K344 as target
   - Choose the signed application binary
   - Configure memory start address according to your linker script
4. Start programming process and wait for completion

Verification

Step 1: Verify Secure Boot Operation

1. Reset the device after programming
2. Observe the LED indicators:
   - GREEN LED blinking slowly indicates successful secure boot
   - RED LED blinking rapidly indicates boot failure or signature verification error

Step 2: Validate Security

1. Test with tampered binary (optional):
   - Modify a single byte in the application binary
   - Sign and program the modified binary
   - Verify that the boot fails (RED LED blinking rapidly)

Troubleshooting

Common Issues

1. **Signature Verification Failure**

   - Ensure correct public key is embedded in the firmware
   - Verify binary hasn't been modified after signing
   - Check HSE configuration settings

2. **Programming Failures**

   - Verify debug connections
   - Ensure device is in correct lifecycle state
   - Check flash protection settings

3. **Key Generation Issues**

   - Verify OpenSSL installation
   - Run Command Prompt as Administrator
   - Check for proper file permissions

Debug Procedures

1. Enable debug output if implemented
2. Verify LED status patterns
3. Use debugger to step through initialization if permitted by security settings

Support Contacts

For additional assistance, contact:

- Technical Support: [support@example.com]
- Security Team: [security@example.com]

Appendix: Command Reference

Key Generation Commands

```
openssl ecparam -name prime256v1 -genkey -noout -out private_key.pem
openssl ec -in private_key.pem -pubout -out public_key.pem
openssl ec -in public_key.pem -pubin -outform DER -out public_key.der
```

Signature Commands

```
openssl dgst -sha256 -sign private_key.pem -out signature.bin binary_file.bin
```