XFS4IoT SP-Dev Workgroup

5 October 2021

XFS4IoT Workgroup Progress

- Framework for 4 devices available
  - Card Reader (released May 2021)
  - Cash Dispenser (July 2021), with full end-to-to security to follow
  - Text Terminal Unit (July 2021)
  - EPP Key Management and Crypto classes (Sept 2021) with Keyboard and PinPad classes to follow
- C# and C++ sample code released, demos on YouTube

XFS4IoT Workgroup Progress

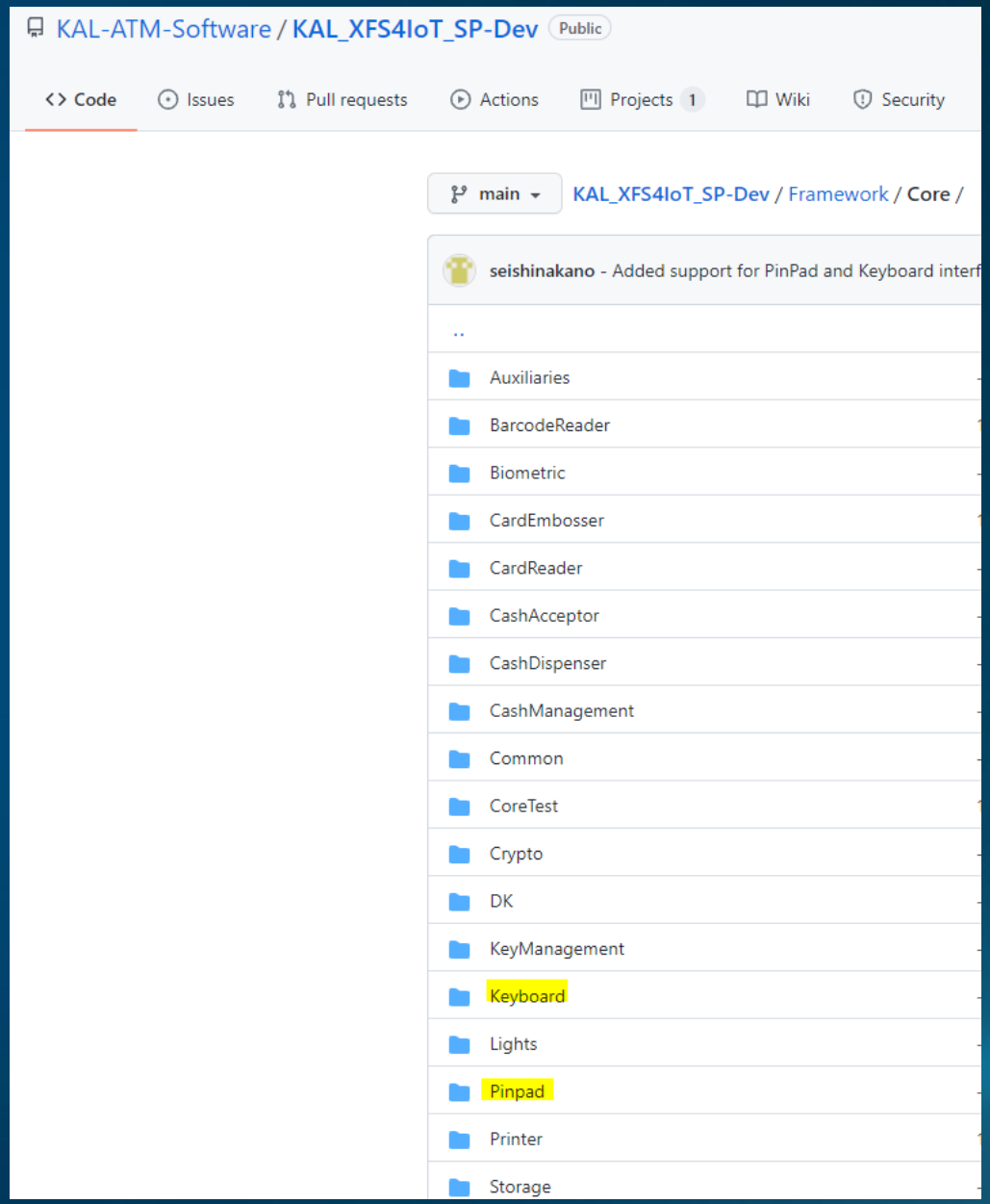
- Framework code is available to:
  - Implement XFS4 SPs
  - Review, test with our samples
  - Write test tools
- Framework code updated regularly

Pinpad and Keyboard classes release

© 2021 KAL ATM Software GmbH (KAL)

Pinpad & Keyboard support

- Pinpad and Keyboard classes now available



Pinpad & Keyboard support

- Details of the changes in the following commit

KAL-ATM-Software / KAL_XFS4IoT_SP-Dev Public

Code | Issues | Pull requests | Actions | Projects 1 | Wiki | Security | Insights | Settings

History for KAL_XFS4IoT_SP-Dev / Framework

Commits on Oct 4, 2021

- Added support for PinPad and Keyboard interfaces framework. ...

seishinakano committed 20 hours ago

KAL-ATM-Software / KAL_XFS4IoT_SP-Dev Public

Code | Issues | Pull requests | Actions | Projects 1 | Wiki | Security | Insights | Settings

- Added support for PinPad and Keyboard interfaces framework.

```
- Updated TextTerminal framework to the latest specification published on the September preview.
- Cancel command supported in the framework.
```

main

v0.0.4-alpha.1

Pinpad & Keyboard support

- Sample also available

KAL-ATM-Software / KAL_XFS4IoT_SP-Dev-Samples Public

Code | Issues 2 | Pull requests | Actions | Projects | Wiki | Security

main | KAL_XFS4IoT_SP-Dev-Samples / Devices /

seishinakano Changed to use E2E public package for C++ CLI

..

SampleCardReader

SampleCashDispenser

SampleEncryptor

SamplePinPad

SampleServerHost

TextTerminalSample

SPs.sln

Pinpad & Keyboard support

- Latest pre-release v.0.0.4-alpha1 for NuGet packages

KAL.XFS4IoT.SP-Dev.Framework.CardReader.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.CardReader.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.CashDispenser.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.CashDispenser.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.CashManagement.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.CashManagement.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.Common.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.Common.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.Core.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.Core.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.Crypto.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.Crypto.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.Keyboard.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.Keyboard.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.KeyManagement.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.KeyManagement.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.Native.EndToEnd.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.PinPad.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.PinPad.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.Server.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.Server.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.ServiceInterfaces.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.ServiceInterfaces.0.0.4-alpha.1.snupkg
KAL.XFS4IoT.SP-Dev.Framework.TextTerminal.0.0.4-alpha.1.nupkg
KAL.XFS4IoT.SP-Dev.Framework.TextTerminal.0.0.4-alpha.1.snupkg

Already supported by KeyManagement and Crypto

- Remote Key Loading with TR34 supported
- Other types of RKL are also fully supported (signature-based, E-RKL…)
- Key Exchange with TR31 supported
- Enabling E2E security feature
- Any device can support Key Management – just requires an HSE

Latest functions supported

- Status and Capabilities
- Data entry
- Pin entry and format Pin
- Secure key entry
- Initialise, load and import key methods
- Get key names and KCV (Key Check Value)

Demo with real EPP

© 2021 KAL ATM Software GmbH (KAL)

Demo with real EPP

Demo video will be available on YouTube.

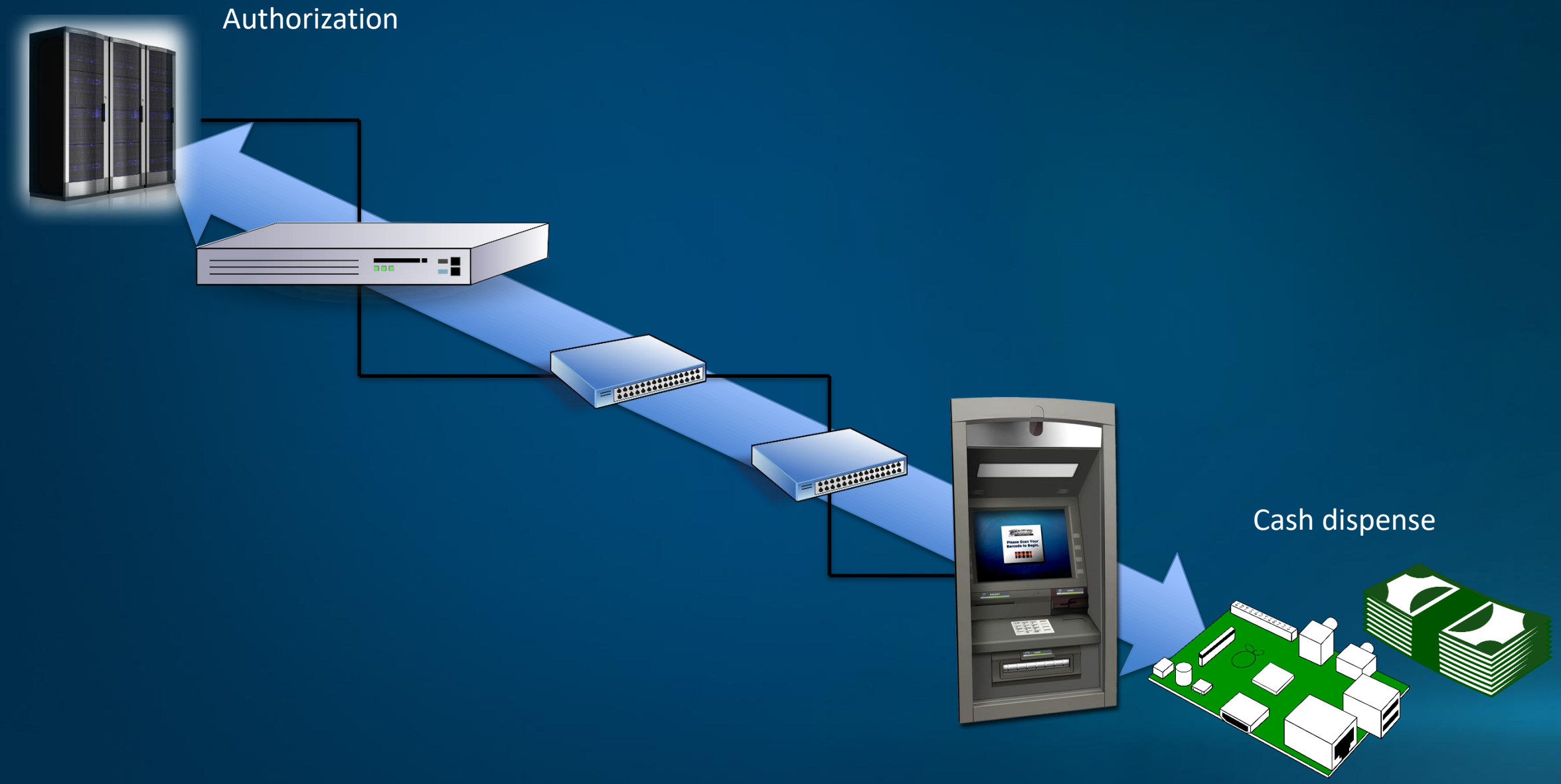
All previous demo videos can be found on the KAL ATM Software YouTube channel:

https://www.youtube.com/user/ATMsoftware/videos

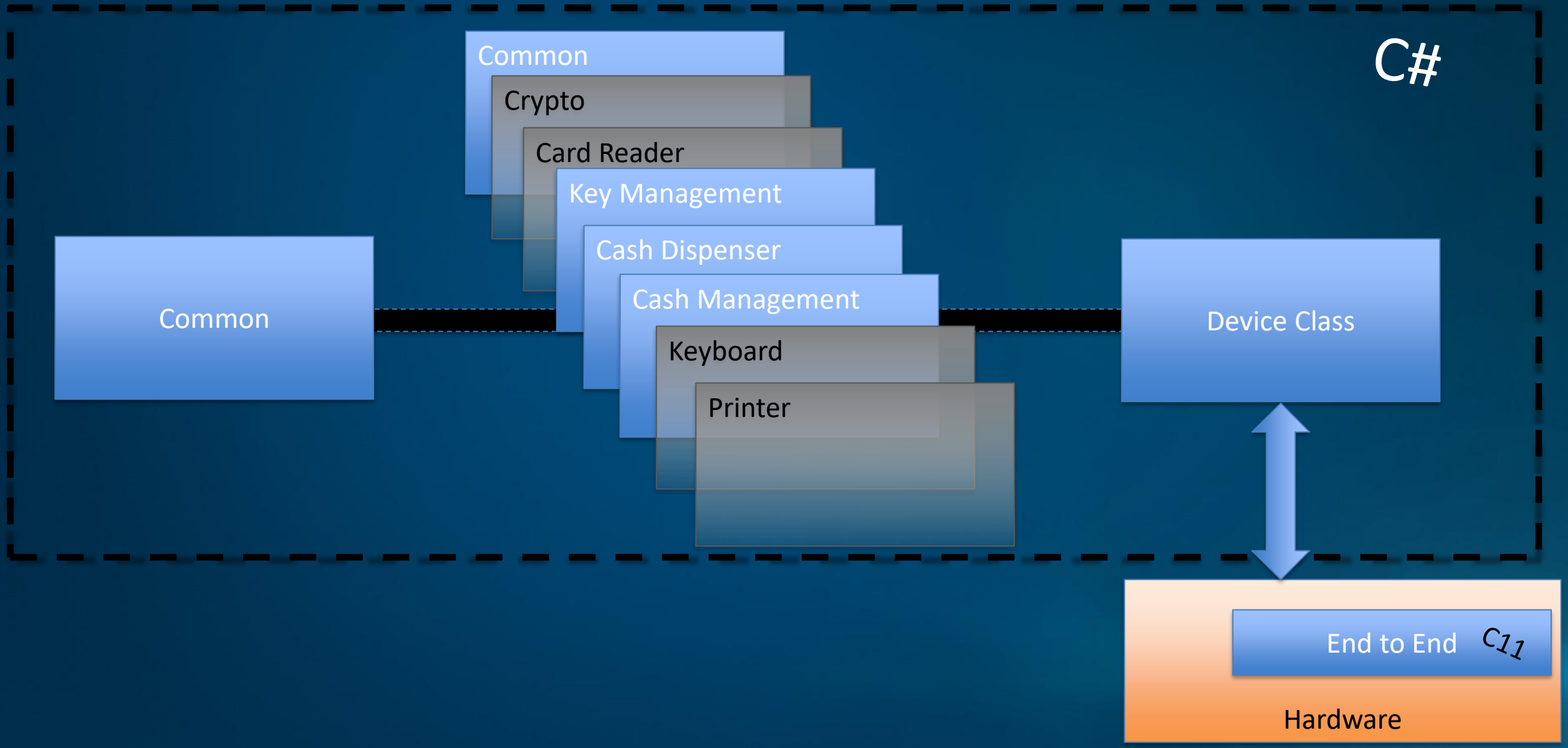
XFS4IoT SP-Dev E2E support

First draft E2E support

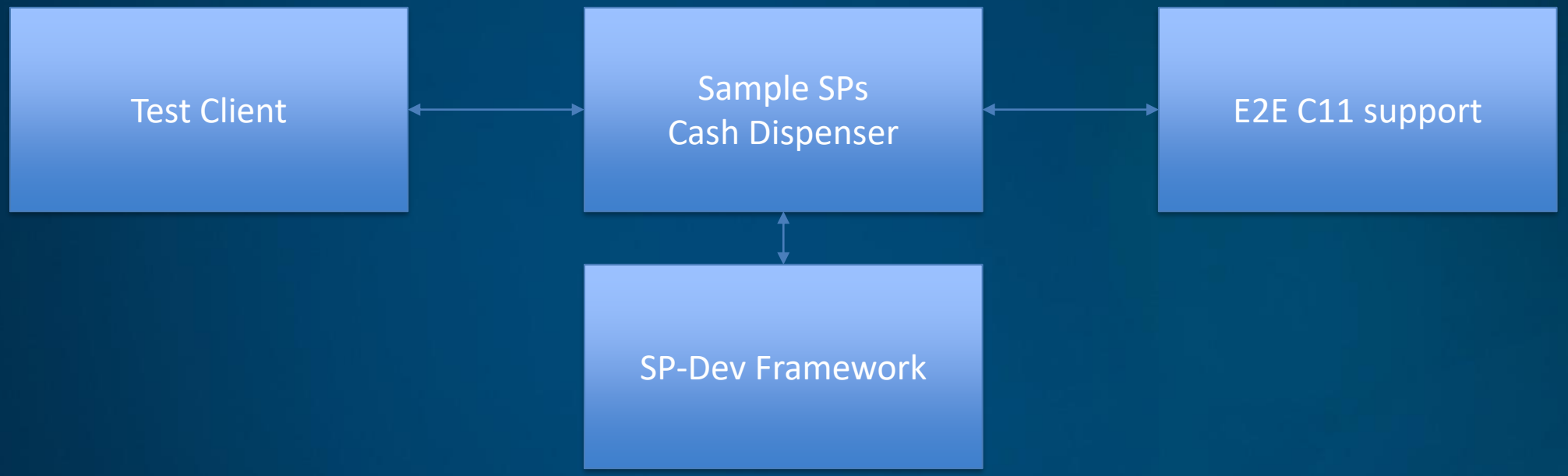
E2E security



Framework with end to end security



Components



TestClient example

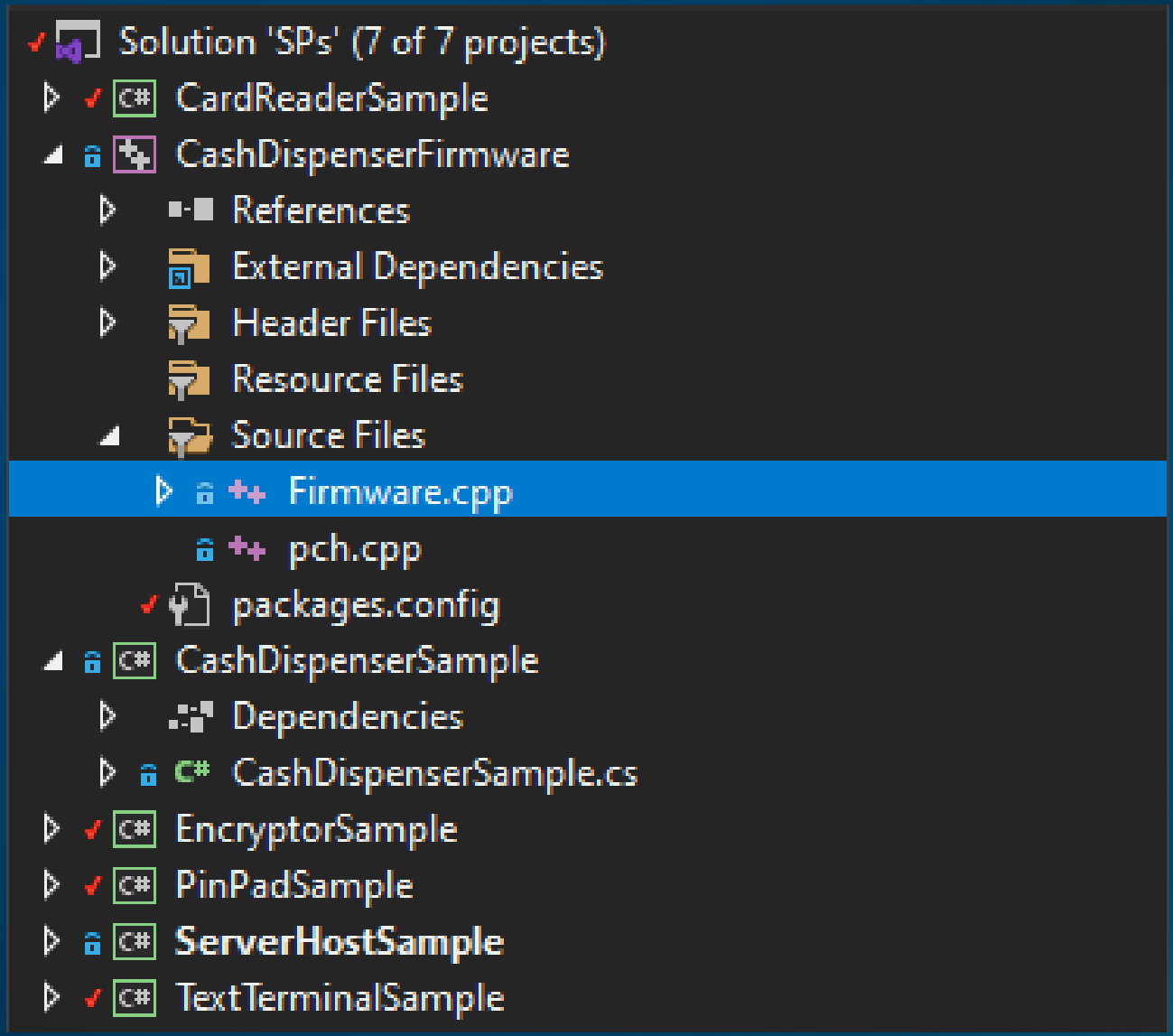
```
5 references | Kit Patterson, 2 hours ago | 1 author, 3 changes
private async Task<string> DoGetCommandNonce()
{
    var command = new GetCommandNonceCommand(RequestId.NewID(),
                                             Payload: new(Timeout: 10_000)
                                             );

    await cashDispenser.SendCommandAsync(command);
    var response = await GetCompletionAsync<GetCommandNonceCompletion>(cashDispenser);
    if (response.Payload.CompletionCode ≠ XFS4IoT.Completions.MessagePayload.CompletionCodeEnum.Success)
        throw new Exception($"GetCommandNonce failed: {response.Payload.CompletionCode}");

    return response.Payload.CommandNonce;
}

7 references | Kit Patterson, 3 days ago | 1 author, 2 changes
private async Task DoDispenseCash(int Amount, string CurrencyID, string Token)
{
    var command = new DispenseCommand(RequestId.NewID(),
                        Payload: new(Timeout: 10_000,
                                     Denomination: new(Currencies: new() { { CurrencyID, Amount } } ),
                                     MixNumber:1,
                                     Token: Token
                                     )
                        );

    await cashDispenser.SendCommandAsync(command);
    var response = await GetCompletionAsync<DispenseCompletion>(cashDispenser);
    if (response.Payload.CompletionCode ≠ XFS4IoT.Completions.MessagePayload.CompletionCodeEnum.Success)
        Logger.LogWarning($"Dispense failed: {response.Payload.CompletionCode}");

}
```

TestClient example

```
4 references | Kit Patterson, 21 hours ago | 1 author, 1 change
private static string MakeToken(string nonce, bool valid)
{
    // 'valid' or invalid HMAC.
    var HMAC = valid ? "CB735612FD6141213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2"
                     : "CB735612FD6141213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F3";

    var tokenBuilder = new System.Text.StringBuilder($"NONCE={nonce},TOKENFORMAT=1,TOKENLENGTH=$$$$,ANOTHERKEY=12345,HMACSHA256={HMAC}");

    // The token length field is fix at four digits to make it easy to calculate.
    // Inject this into the string.
    var len = $"{tokenBuilder.Length:X4}";
    tokenBuilder = tokenBuilder.Replace("$$$$", len);

    string token = tokenBuilder.ToString();
    return token;
}
```

Don’t do this!!!

Sample service



Sample cash dispenser

```
0 references | Kit Patterson, 21 hours ago | 2 authors, 5 changes
public async Task<DispenseResult> DispenseAsync(IDispenseEvents events, DispenseRequest dispenseInfo, Cancellatior
{
    if (dispenseInfo.E2EToken is null)
        return new DispenseResult(MessagePayload.CompletionCodeEnum.InvalidToken, dispenseInfo.Values, LastDispens

    if ( !Firmware.VerifyAndDispense(dispenseInfo.E2EToken) )
    {
        return new DispenseResult(MessagePayload.CompletionCodeEnum.InvalidToken, dispenseInfo.Values, LastDispens
    }
```

Sample ‘firmware’

```
bool KAL::XFS4IoTSP::CashDispenser::Sample::Firmware::VerifyAndDispense(System::String^ Token)
{
    // Convert the .net UTF16 string into a native UTF8 string.
    pin_ptr<const wchar_t> pinnedToken = PtrToStringChars(Token);
    wstring wideToken = pinnedToken;

    wstring_convert<codecvt_utf8<wchar_t>> utf8Converter;
    string utf8Token;
    try
    {
        utf8Token = utf8Converter.to_bytes(wideToken);
    }
    catch (range_error const &)
    {
        cout << "Error: Invalid token. Conversion to UTF8 failed after " << dec << utf8Token.size() << " characters:\n";
        for (auto c : utf8Token)
            cout << hex << showbase << c << '\n';
    }

    // Check that the token is valid.
    // Include null in token (buffer) size.
    auto tokenValid = ValidateToken(utf8Token.c_str(), utf8Token.size()+1);
    if (!tokenValid)
        return false;

    return true;
}

System::String^ KAL::XFS4IoTSP::CashDispenser::Sample::Firmware::GetCommandNonce() { ... }

void KAL::XFS4IoTSP::CashDispenser::Sample::Firmware::ClearCommandNonce() { ... }
```

Core firmware check

```
/// <summary>
/// Validate that a token has a valid format.
/// </summary>
/// <description>
/// Takes a token string and reports if it is valid or not. If the token is valid then it is safe
/// to proceed with the operation protected by the token. If not, the operation should fail with an
/// error.
/// The token still is assumed to be unsafe and may have been passed by an attacker. Everything
/// possible will be done to avoid invalid behaviour due to a hostile token.
/// </description>
///
/// <param name="Token">Null terminated token string</param>
/// <param name="TokenSize">Token buffer size, including null</param>
/// <returns>true or false</returns>
bool ValidateToken(char const* const Token, size_t TokenSize)
{
    LogV("ValidateToken( Token=\"%.1024s\", TokenSize=%d )", Token, TokenSize);

    // Parameter checking.
    // Consider the token to be an untrusted value, so maximum validity checking.
    // Null token
    if (Token == NULL)
    {
        Log("ValidateToken: Null token ⇒ false");
        return false;
    }

    size_t TokenStringLength = strlen(Token) + 1;       // Plus null
    // Zero length string.
    if (TokenStringLength == 1)
    {
        Log("ValidateToken: Empty token ⇒ false");
        return false;
    }
    // Buffer length and string size don't match
    if (TokenStringLength ≠ TokenSize)
    {
        Log("ValidateToken: TokenSize didn't match token length ⇒ false");
        return false;
    }
```

Firmware functions to implement

```
/// ...
extern C_LINKAGE void NewNonce( char const ** Nonce );

/// ...
extern C_LINKAGE bool CompareNonce(char const* const CommandNonce, size_t NonceLength);

/// ...
extern C_LINKAGE void ClearNonce();

/// ...
extern C_LINKAGE bool CheckHMAC(char const *const Token, unsigned int TokenLength, unsigned char const* const TokenHMAC);

/// ...
extern C_LINKAGE void FatalError(char const* const Message);

/// ...
extern C_LINKAGE void Log(char const* const Message);
```

Test client

```
04:50:34.2882 (00:00:20.8046100):  25:GetCommandNonceCommand
04:50:34.2898 (00:00:20.8061785):  25:Acknowledge
04:50:34.2906 (00:00:20.8070408):  25:GetCommandNonceCompletion
04:50:34.2919 (00:00:20.8083508): Nonce : 2
04:50:34.2924 (00:00:20.8088389): Dispense (valid token)
04:50:34.2932 (00:00:20.8095898): Token: NONCE=2,TOKENFORMAT=1,TOKENLENGTH=0083,ANOTHERKEY=12345,HMACSHA256=CB735612FD6141213C2827F
B5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2
04:50:34.3160 (00:00:20.8323777):  26:DispenseCommand
04:50:34.3439 (00:00:20.8603445):  26:Acknowledge
04:50:35.4792 (00:00:21.9956534):  26:DispenseCompletion
04:50:35.4868 (00:00:22.0032205):  27:PresentCommand
04:50:35.5002 (00:00:22.0166561):  27:Acknowledge
04:50:36.5835 (00:00:23.0999570):  27:PresentCompletion
04:50:36.5842 (00:00:23.1006132): Dispense (Stale token)
04:50:36.5846 (00:00:23.1010338): Token: NONCE=2,TOKENFORMAT=1,TOKENLENGTH=0083,ANOTHERKEY=12345,HMACSHA256=CB735612FD6141213C2827F
B5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2
04:50:36.5853 (00:00:23.1017263):  28:DispenseCommand
04:50:36.5880 (00:00:23.1044479):  28:Acknowledge
04:50:37.6065 (00:00:24.1229289):  28:DispenseCompletion
04:50:37.6085 (00:00:24.1249283): Dispense (Invalid HMAC)
04:50:37.6095 (00:00:24.1259153): Invalid HMAC: NONCE=2,TOKENFORMAT=1,TOKENLENGTH=0083,ANOTHERKEY=12345,HMACSHA256=CB735612FD614121
3C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F3
04:50:37.6109 (00:00:24.1273395):  29:DispenseCommand
04:50:37.6171 (00:00:24.1335197):  29:Acknowledge
04:50:37.6231 (00:00:24.1395400):  29:DispenseCompletion
04:50:37.6246 (00:00:24.1410308): Dispense failed: InvalidToken
04:50:37.6255 (00:00:24.1419278): Dispense (Invalid nonce)
04:50:37.6264 (00:00:24.1427922): Invalid nonce: NONCE=FFFF,TOKENFORMAT=1,TOKENLENGTH=0086,ANOTHERKEY=12345,HMACSHA256=CB735612FD61
41213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2
04:50:37.6272 (00:00:24.1436531):  30:DispenseCommand
04:50:37.6305 (00:00:24.1469156):  30:Acknowledge
04:50:37.6341 (00:00:24.1505328):  30:DispenseCompletion
04:50:37.6356 (00:00:24.1519863): Dispense failed: InvalidToken
04:50:37.6359 (00:00:24.1523171): Done
```

Status

- Complete
  - Added firmware C library support (including unit tests.)
  - Framework support for Common.GetCommandNonce, Common.ClearCommandNonce, and CashDispenser.Dispense token commands.
  - Sample SP implementation using ‘firmware’ code and dispenser implementation
  - (Command line) test client showing dispense and present sequence

To do

- Read and track token keys – such as dispense amount
- Enforce dispense values
- Present should delete the nonce/invalidate tokens
- Support for generating response tokens
- GetPresentStatus token support
- UI test client

Not included

- Key handling – KeyManagement, but needs hardware support
- Cryptography – Must be done in hardware
- Random number or persistent storage – needs to be done in hardware

XFS4 Specification: roadmap update and September preview

© 2021 KAL ATM Software GmbH (KAL)

XFS4IoT specification

- September preview published
- Online version and downloaded PDF available (link)

XFS4IoT specification

- Roadmap:
  - October:
    - Code freeze
    - Document generation
    - Final review period by CEN committee members
  - Mid-December:
    - Final validation
    - Release process with CEN / Preview updated

XFS4IoT specification : Latest updates

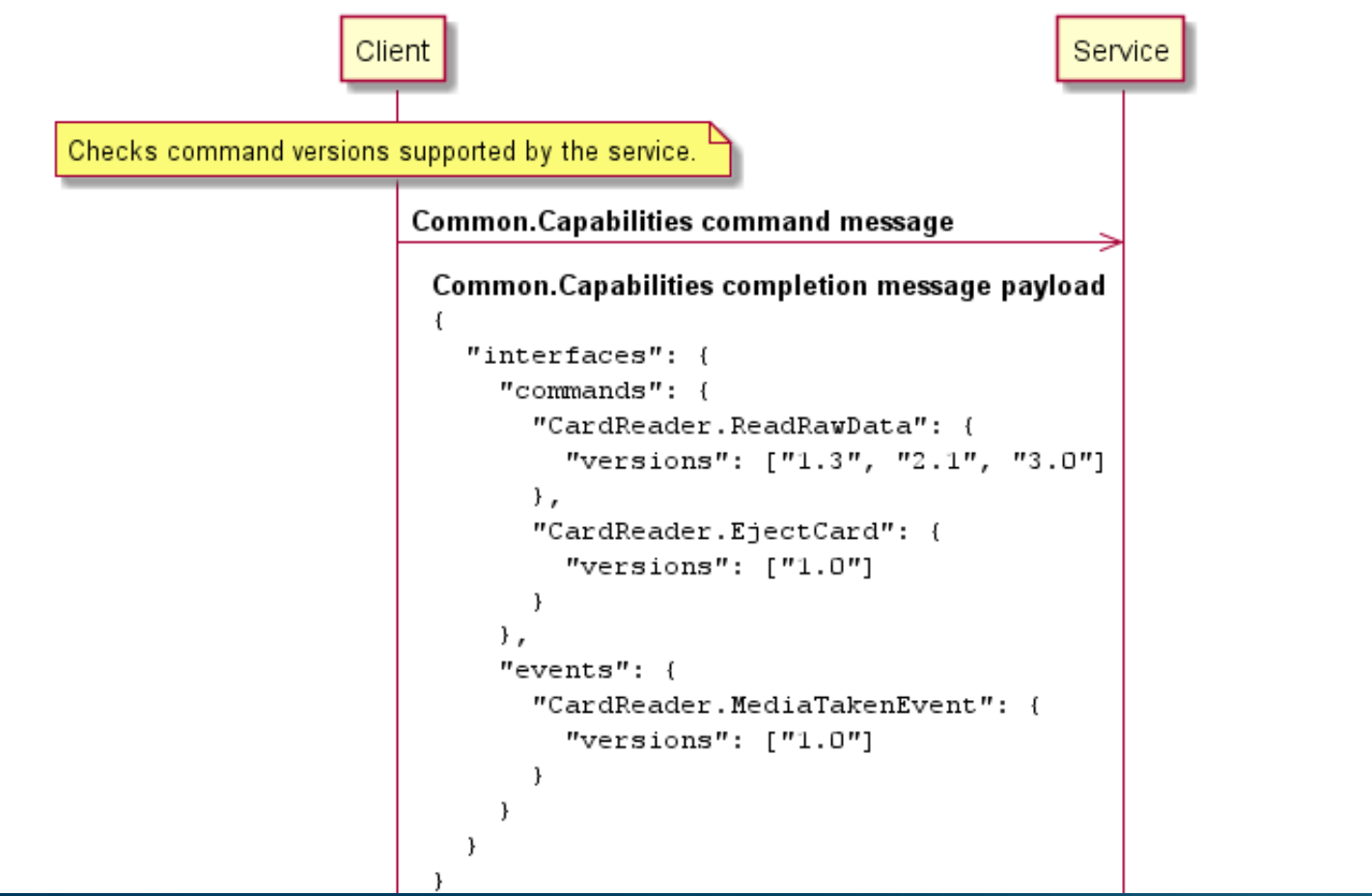
- New card reader interface:
  - Dispensing capabilities
  - New “Move” command
- New Storage Interface
  - To be used with other device interfaces which requires storage
  - Bins and cash unit storage handling

XFS4IoT specification : Latest updates

- Service discovery own interface:
  - Move out of Common
- New Lights and Auxiliaries interfaces
  - Doors
  - Alarm
  - Generic sensors…

XFS4IoT specification : Latest updates

- Versioning information
  - Capabilities command
  - Per command
  - Per event



What’s next?

© 2021 KAL ATM Software GmbH (KAL)

What's next?

- Updating the current framework with the latest specification changes from the September preview
- Support more classes:
  - Printer
  - Vendor Mode/Vendor Application
  - Auxiliaries

→ *Everything we need to support a complete Cash Out ATM…*

What’s next?

- Full E2E process on real hardware
- Proof of concept on small devices using a TPM chip as HSE
- Support workgroup members with implementation using the framework
- Guest speakers

Next call

MS Teams

- First Tuesday of each month at 1300 UK time

Next call: 2nd November 2021, 1300 UK, 0900 US EST, 2200 Tokyo time

(Note: **US** changes clocks on the 7th November, so the call will be back to the 0800 EST slot for the US from December).