AdaTLS – Project Summary
Summer Term 2015

Jiaqi Weng

October 15, 2015

1 Introduction

The AdaTLS project is including Ada Language Learning and TLS protocol working.

2 Ada Basic Learning

2.1 Installing GNAT and Ada Environment

For your home machine you will need a compiler and an environment for creating, compiling, and running programs.

Follow Next Article,
`http://www.radford.edu/~nokie/classes/320/compileInstall.html`

2.2 Debugging Ada programming on GPS

GPS using GDB to debug Ada programming,

Follow Next Article,
`http://docs.adacore.com/gps-docs/users_guide/_build/html/debugging.html`
For Mac OS, GDB need Codesigning,

Follow Next Article,
`https://gcc.gnu.org/onlinedocs/gcc-4.9.2/gnat_ugn/Codesigning-the-Debugger.html#Codesigning-the-Debugger`

2.3 Ada Programming Learning

Basic Contents and Exercise,

- Fundamentals of Ada
- Statements
- Procedures, functions and packages
- Defining new data types
- Composite data types
- Exceptions

Follow Next Article,
`http://www.adaic.org/resources/add_content/docs/craft/html/contents.htm`

3 Ada Crypto Library

3.1 Prerequisites

GNAT-4.8 (recommended version: 4.9)
GNU Make (recommended version: 4)
gcov (recommended version: 4.8.3; optional for testing only)
lcov (recommended version: 1.10; optional for testing only)

3.2 To build and test

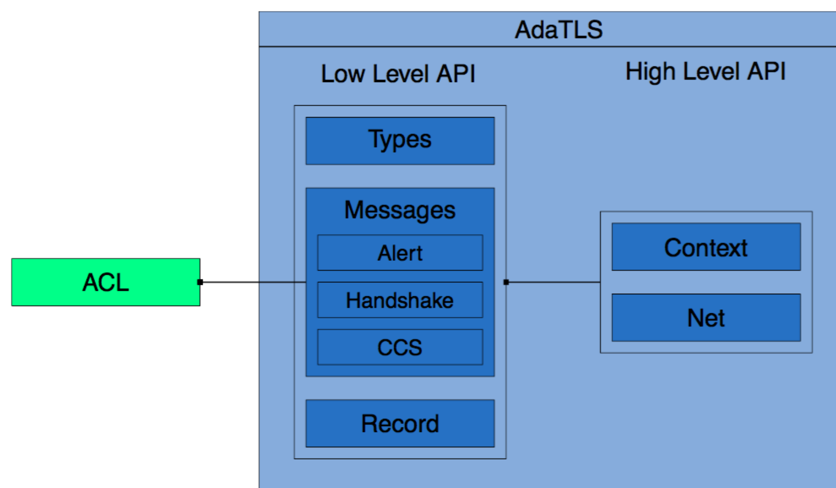
make
make acltest
cd test
./test-tests

3.3 To build the PDF documentation

(location: doc/):
make docu

4 TLS Protocol and Handshake

4.1 Packages



- Types
  - Elementary Types
  - Declaration of cipher suites

- Context
  - Storage of all parameters of the connection
- Messages
  - Interface
  - Routines for sending and receiving messages
- Record
  - Processing of application data
  - Encryption and decryption
  - Calculation of MACs
- Net
  - Initialisation of handshakes
  - Send and Receive application data

4.2 Constraint Error

Running with a constraint error

```
Parse_Server_Message
HType: HT_SERVER_HELLO_DONE
Client_Handshake STATE: STATE_CLIENT_KEY_EXCHANGE
 1024
HType: HT_CLIENT_KEY_EXCHANGE
Client_Handshake STATE: STATE_CERTIFICATE_VERIFY
Client_Handshake STATE: STATE_CLIENT_CHANGE_CIPHER_SPEC
Client_Handshake STATE: STATE_CLIENT_FINISHED

Client_Handshake STATE: STATE_SERVER_CHANGE_CIPHER_SPEC
Parse_Server_Message

raised CONSTRAINT_ERROR : tls-handshake-client_key_exchange.adb:356
     length check failed
CH-E: ERROR_READ_FAILED
Error while performing handshake: ERROR_READ_FAILED
```

Seems like P is too big.

```
    -- Get the encrypted premaster secret
               Context.Read_Buffer.Read(Enc_Premaster);
               P_RSA.Set_Private_Key(RSA_PK.M.Element,
                        RSA_PK.D.Element,
                        RSA_PK.P.Element,
                        RSA_PK.Q.Element,
                        RSA_PK.Phi.Element,
                        Private_Key);
```

There are two types of procedure $Set_Private_Key$

```
    procedure Set_Private_Key(N    : in RSA_Number;
                              D    : in RSA_Number;
                              P    : in RSA_Number;
                              Q    : in RSA_Number;
                              Phi : in RSA_Number;
                              Private_Key : out Private_Key_RSA) is

   procedure Set_Private_Key(N    : in Big_Unsigned;
                              D    : in Big_Unsigned;
                              P    : in Big_Unsigned;
                              Q    : in Big_Unsigned;
                              Phi : in Big_Unsigned;
                              Private_Key : out Private_Key_RSA) is
```

The solution is to convert arguments to Big_Unsigned anyway, but it is just for skipping the error not solved.

```
      Private_Key.N := To_Big_Unsigned(N);
      Private_Key.D := To_Big_Unsigned(D);
      Private_Key.P := To_Big_Unsigned(P);
      Private_Key.Q := To_Big_Unsigned(Q);
      Private_Key.Phi := To_Big_Unsigned(Phi);

      Private_Key.N := N;
      Private_Key.D := D;
      Private_Key.P := P;
      Private_Key.Q := Q;
      Private_Key.Phi := Phi;
```

And it works on Client

```
Client_Handshake STATE: STATE_CLIENT_FINISHED
Client_Handshake STATE: STATE_SERVER_CHANGE_CIPHER_SPEC
Parse_Server_Message
Client_Handshake STATE: STATE_SERVER_FINISHED
Parse_Server_Message
HType: HT_FINISHED
Received verify data:
0668A02BC12A7FE10FDF02EB
Client_Handshake STATE: STATE_APPLICATION_DATA
Loop:  1
-- START Received message:  16 bytes --
TLS World
-- END Received message --
Loop:  2
Parsing Alert Message:  0
-- START Received message:  0 bytes --
-- END Received message --
Error while receiving server response: CONNECTION_CLOSED
```

and Server

```
Parse_Client_Message

HType: HT_FINISHED
Received verify data:
```

```
9B233049654F9A268F8ADC52

Server_Handshake STATE: STATE_SERVER_CHANGE_CIPHER_SPEC

Server_Handshake STATE: STATE_SERVER_FINISHED

Server_Handshake STATE: STATE_APPLICATION_DATA
Received message: 59 bytes
GET / HTTP/1.1
Host: 127.0.0.1:4443
Connection: close
```

Code Difference

```
diff --git a/src/tls-handshake-client_key_exchange.adb b/src/tls
index 22f7a11..4cf689b 100644
--- a/src/tls-handshake-client_key_exchange.adb
+++ b/src/tls-handshake-client_key_exchange.adb
@@ -351,11 +351,11 @@ package body TLS.Handshake.Client_Key_Exchange is
          begin
             -- Get the encrypted premaster secret
             Context.Read_Buffer.Read(Enc_Premaster);
-            P_RSA.Set_Private_Key(RSA_PK.M.Element,
-                                  RSA_PK.D.Element,
-                                  RSA_PK.P.Element,
-                                  RSA_PK.Q.Element,
-                                  RSA_PK.Phi.Element,
+            P_RSA.Set_Private_Key(P_RSA.Big.Utils.To_Big_Unsigned(RSA_PK.M.Element),
+                                  P_RSA.Big.Utils.To_Big_Unsigned(RSA_PK.D.Element),
+                                  P_RSA.Big.Utils.To_Big_Unsigned(RSA_PK.P.Element),
+                                  P_RSA.Big.Utils.To_Big_Unsigned(RSA_PK.Q.Element),
+                                  P_RSA.Big.Utils.To_Big_Unsigned(RSA_PK.Phi.Element),
                                   Private_Key);

 --                P := P_RSA2048.Big.Utils.To_Big_Unsigned(P1_2048);
```

4.3 Install and run the server and client on different machine

1. install the test_server and certificates on one machine.
–bin
——test_server
–keys
——cert_chain.txt
——privatekey.pem
——trust_info.txt
...

2. running test_server with 2 parameters -a and -p.
-a is domain name or ip address, -p is port.
for example,

```
./test_server -a dblsec11.medien.uni-weimar.de -p 4443
Or
./test_server -a 141.54.159.131 -p 4443
```

Result of running

```
jweng@dblsec11:~/bin$ ./test_server -a dblsec11.medien.uni-weimar.de -p 4443
Bind at port  4443


-------------------------
Starting TLS Server
Listening on socket  4443
Server side socket  4443
```

3. call test_client with the same arguments on another machine.

```
./test_client -a dblsec11.medien.uni-weimar.de -p 4443
Or
./test_client -a 141.54.159.131 -p 4443
```

4. We can get the successful handshake result.

```
Client_Handshake STATE: STATE_CLIENT_FINISHED
Client_Handshake STATE: STATE_SERVER_CHANGE_CIPHER_SPEC
Parse_Server_Message
Client_Handshake STATE: STATE_SERVER_FINISHED
Parse_Server_Message
HType: HT_FINISHED
Received verify data:
0668A02BC12A7FE10FDF02EB
Client_Handshake STATE: STATE_APPLICATION_DATA
Loop:  1
-- START Received message:  16 bytes --
TLS World
-- END Received message --
Loop:  2
Parsing Alert Message:  0
-- START Received message:  0 bytes --
-- END Received message --
Error while receiving server response: CONNECTION_CLOSED
```

5 Certification Path Validation

5.1 Summary

1. Definition RFC5280:
https://tools.ietf.org/html/rfc5280#section-6

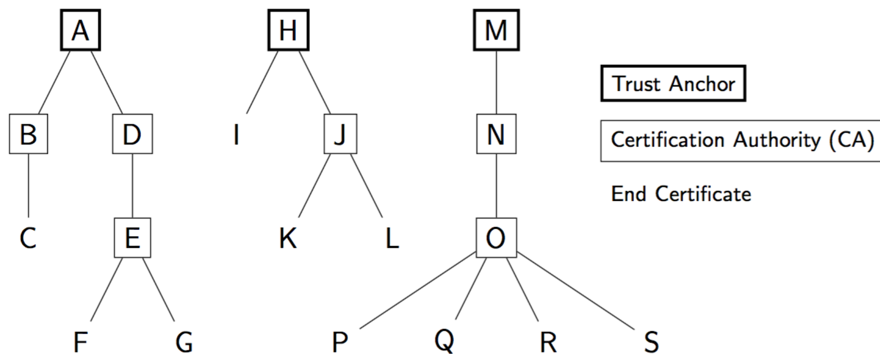
```
Certificate ::= SEQUENCE {
       tbsCertificate            TBSCertificate,
       signatureAlgorithm        AlgorithmIdentifier,
       signatureValue            BIT STRING }

TBSCertificate issuer  ::=  SEQUENCE {
     version          [0]  Version DEFAULT v1,
```

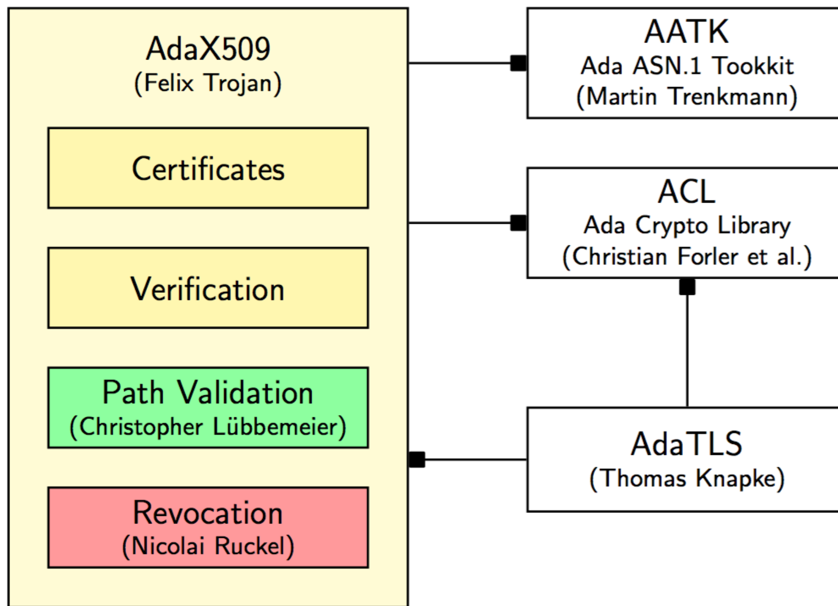
```
    serialNumber         CertificateSerialNumber,
    signature            AlgorithmIdentifier,
    issuer               Name,
    validity             Validity,
    subject              Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                         -- If present, version MUST be v2 or v3
    subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                         -- If present, version MUST be v2 or v3
    extensions      [3]  Extensions OPTIONAL
                         -- If present, version MUST be v3 --  }
```

2. Public Key Infrastructure (PKI)

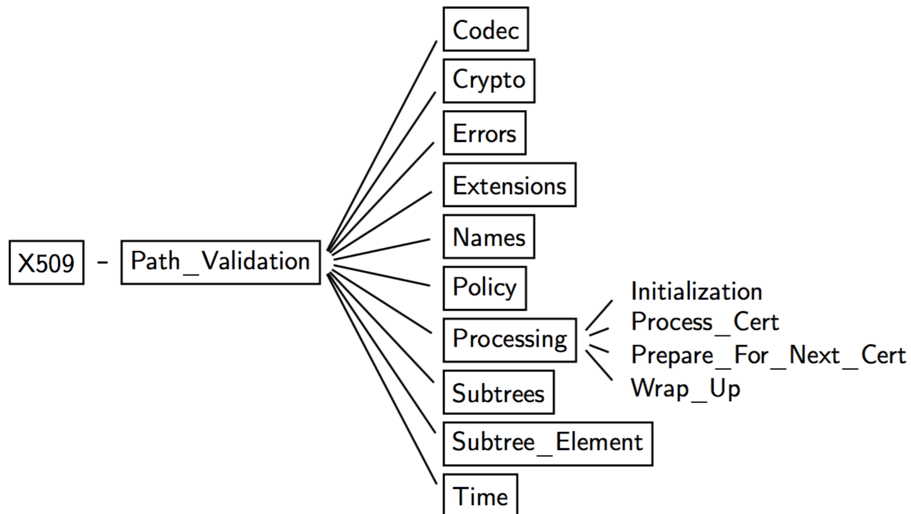
A public key infrastructure (PKI) is a set of hardware, software, people, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates[1] and manage public-key encryption.
- Wikipedia



3. Module Structure



4. X509 Path Validation Structure



5.2 Unit Test Cases

The unit test cases of PKI is from
`http://csrc.nist.gov/groups/ST/crypto_apps_infra/pki/pkitesting.html`

Test Description:
`http://csrc.nist.gov/groups/ST/crypto_apps_infra/documents/PKITS.pdf`

Test Data Downloads:

http://csrc.nist.gov/groups/ST/crypto_apps_infra/documents/PKITS_data.zip

Types:

- Signature Verification.
- Validity Periods
- Verifying Name Chaining
- Basic Certificate Revocation Tests
- Verifying Paths with Self-Issued Certificates
- Verifying Basic Constraints
- Key Usage
- Certificate Policies
- Require Explicit Policy
- Policy Mappings
- Inhibit Policy Mapping
- Inhibit Any Policy
- Name Constraints
- Distribution Points
- Delta-CRLs

5.3 x509_tool

The tool is used to create certificate with a property definition file and verify the certificate. And we add new functions to create certificate path.

Problem:
The tool need to hard code for add additional extensions to certificate.

```
declare
   use Extensions;
   Item: Extensions.Sequence;
begin
     Item.Append(Extension'(c_extnID=> RFC5280.Implicit.
    id_ce_basicConstraints,
          c_critical  => Bool_True,
          c_extnValue => To_Octet_String ("30030101FF")));
     Item.Append(Extension'(c_extnID => RFC5280.Implicit.
    id_ce_keyUsage,
          c_critical  => Bool_True,
          c_extnValue => To_Octet_String ("03020106")));
     Certificate.c_tbsCertificate.c_extensions.Set_Element( Item );
end;
```

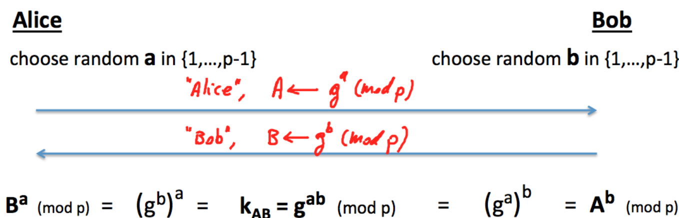
The way to add extension need to using asn.1 javascript decoder to get the code of extension.
The decoder tool is in the folder src/adax509/data/tools/jsdecoder/index.html
Or it could be got online under next site.
https://lapo.it/asn1js/

6 Ephemeral Diffie-Hellman

6.1 Summary

1. Diffie-Hellman

Fix a large prime p (e.g. 600 digits)
Fix an integer g in {1, ..., p}



2. PFS(Perfect Forward Secrecy)

Problem:
If all the traffic has been recorded by an attacker, they have Random Number #1 and #2 as they are sent in plain text, along with the Pre-Master Secret encrypted with the server public key. Once the attacker has the server private key, they can decrypt the Pre-Master Secret and generate the Master Secret to decrypt the session data.

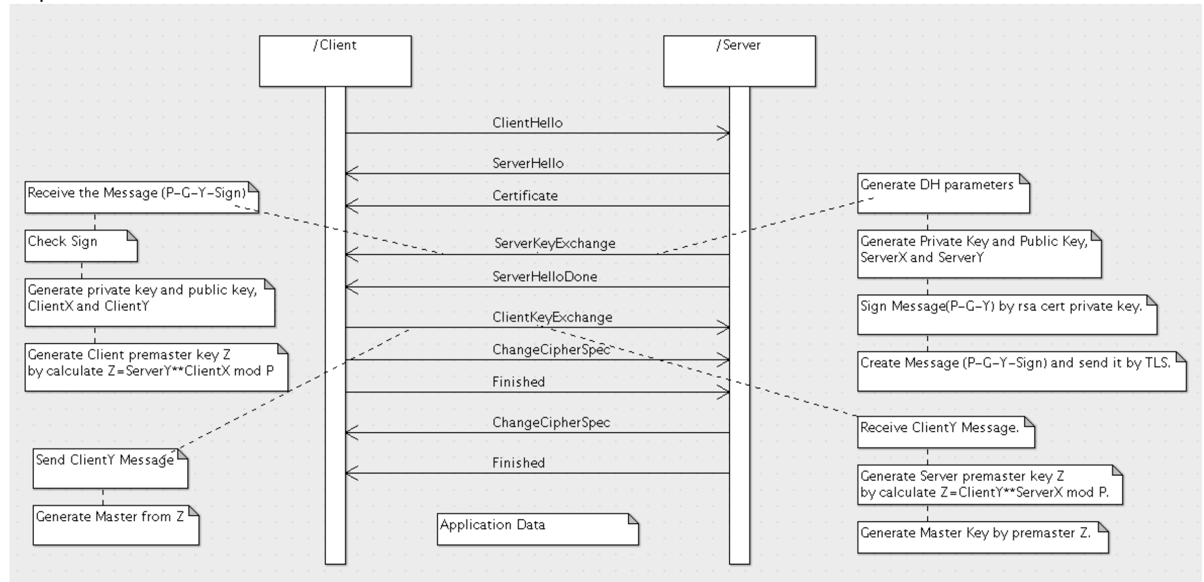
Solution:
To enable PFS, the client and the server have to be capable of using a cipher suite that utilises the Diffie-Hellman key exchange. Importantly, the key exchange has to be ephemeral. This means that the client and the server will generate a new set of Diffie-Hellman parameters for each session. These parameters can never be re-used and should never be stored, the ephemeral part, and that's what offers the protection going forwards.
Even if the attacker managed to compromise this shared secret somehow, it would only compromise that particular session. No previous or future sessions would be compromised.

Reference:
https://scotthelme.co.uk/perfect-forward-secrecy/

3. Ephemeral Diffie-Hellman

Uses temporary, public keys. Each instance or run of the protocol uses a differ-

ent public key. The authenticity of the servers temporary key can be verified by checking the signature(RSA) on the key. Because the public keys are temporary, a compromise of the serverś long term signing key does not jeopardise the privacy of past sessions.
Implementation:



and
ClientZ = ServerY**ClientX mod P = g ** (ServerX * ClientX) mod P
ServerZ = ClientY ** ServerX mod P = g ** (ClientX * ServerX) mod P

6.2 About Specification of DH parameters

Problem:
How often should the DH parameter p and g update on Server?

Investigation:
The Specification of TLS 1.2 is on
`https://tools.ietf.org/html/rfc5246#section-8.1.2`
There is notice for Diffie-Hellman Key Exchange in the document.
"
Note: Diffie-Hellman parameters are specified by the server and may be either ephemeral or contained within the server's certificate.
"
But there is no details about how to make it ephemeral.

On another specification of rfc2631 defines how to generate DH parameters.
`https://www.ietf.org/rfc/rfc2631.txt`
but there is no contents for when and how often to update the DH parameters on Server.

And there are also other specification documentation including Diffie-Hellman on-

line.
https://tools.ietf.org/html/rfc2409
But still cannot find some about when and how often to update the DH parameters on Server.

And then there are some discussions about it.
http://crypto.stackexchange.com/questions/10820/is-prime-regeneration-necessary-for-every-
http://crypto.stackexchange.com/questions/1963/how-large-should-a-diffie-hellman-p-be/1964#1964
http://crypto.stackexchange.com/questions/8947/why-use-variable-p-q-g-for-diffie-hellman

Mostly the dh parameters should not changed in every sessions, and parameters should be public values and written in to standards.
There are some examples online.
http://www.rfc-editor.org/rfc/rfc3526.txt
http://www.ietf.org/rfc/rfc5114

Solution:
1, keep the parameter p and q static, due to the specification need key-pair is ephemeral on every session and it is not necessary to change the parameters.

2, It is found a solution in rfc4419 (Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol)
https://www.ietf.org/rfc/rfc4419.txt
the solution in this document is
"
The server keeps a list of safe primes and corresponding generators that it can select from. A prime p is safe if $p = 2q + 1$ and q is prime. New primes can be generated in the background.
"
And an easy way is using tools to create a dh parameter file regularly on Sever backend e.g. openssl, and load the parameter from the file in every session.
The openssl dh command is on
https://www.openssl.org/docs/manmaster/apps/dhparam.html

Example:

```
jack:bin wejaq$ openssl dhparam -outform PEM -out dHParam.pem 1024 -text -C
Generating DH parameters, 1024 bit long safe prime, generator 2
This is going to take a long time
...........+..............+..........+.........................................................
.................................................................+.............................
............+....................................................................+.............
.........................................................+...................+...+.............
...........+..............+.............................................................+......
...............................................................................................
.....................+.....................................................................+...
..............................+..................................................+.............
............................++*++*++*
#ifndef HEADER_DH_H
#include <openssl/dh.h>
#endif
DH *get_dh1024()
        {
        static unsigned char dh1024_p[]={
                0xED,0xEF,0x76,0x85,0x7A,0x18,0xF2,0x00,0xBE,0x19,0xE0,0x48,
                0x70,0x8A,0x61,0x52,0xBF,0x5A,0xE5,0xBB,0xF0,0x14,0x67,0xB5,
                0xC1,0x18,0x5A,0x3D,0xAE,0x79,0x57,0x55,0xA4,0xA3,0x6C,0xFF,
                0x56,0x18,0x4E,0x60,0xB7,0x48,0xF8,0x32,0x3D,0xAF,0xA6,0x91,
                0xFB,0xBA,0xB7,0xD6,0x5D,0xF1,0x05,0x21,0x68,0x57,0x3D,0xB2,
                0x4B,0x9B,0xEC,0x8C,0xFC,0xF5,0xB9,0xFB,0xA1,0x7D,0xD3,0x47,
                0xB6,0x84,0xF9,0xAB,0x64,0xCE,0xD9,0xE0,0xE0,0x77,0x94,0x1B,
                0xC1,0x7C,0xD4,0x63,0x25,0x11,0x62,0x4C,0x25,0x2B,0x3D,0x91,
                0x1E,0xC3,0x0B,0x1B,0x07,0x3B,0xBC,0xA9,0x00,0xDC,0xF1,0x4E,
                0x25,0xF3,0xF4,0x79,0xE3,0xA0,0x5D,0x9C,0x87,0x44,0xFF,0xF3,
                0x0D,0x4D,0xB1,0x49,0x1F,0xD3,0xB2,0x2B,
                };
```