

# TLS AND CRYPTOGRAPHY

Hardening TLS using ATECC508A

#### **Prerequisites**

- Hardware Prerequisites
  - AWS Zero Touch Secure Provisioning Kit
- Software Prerequisites
  - Windows 7 or above

#### Introduction

The world we live in is a connected world. Today we rely on our phones, computers and soon IoT devices to communicate, buy goods, travel and work. It is expected that connected devices or IoT devices, increase exponentially in the near future. The fact is that there are more phones than people today. The number of smart phones is measure by the billions and is increasing in a fast pace.

All these devices that are connected to the internet have one thing in common – They rely on the protocol called TLS (Transport Layer Security) to protect their information in transit.

TLS is a cryptographic protocol designed to provide secure communication over an insecure infrastructure. This means that, if this protocol is properly deployed, you can open a communication channel to an arbitrary service on the internet and be reasonably sure that you're talking to the correct server, and exchange information safely knowing that your data won't fall into the wrong hands and that it will be received intact.

These is not the case in the real world. Poorly designed systems along with software bugs can open a back door to an attacker. Aside from this, the simplicity of the RSA algorithm (which is widely used in most of the systems running TLS), has known weaknesses, such as the Private Key being stored in software. Anyone with the access to the corresponding server's private key can decrypt the communication between the client and it. This type of attack does not need to happen in real time. An attacker could stablish a long term operation and record all the encrypted traffic and wait until he obtains the Key. After the Key has been compromised, it's possible to decrypt all previously recorded traffic.

As we will see in the following sections of this document ATECC508 will serve as a secure hardware storage device to store Keys. It will also provide the hardware acceleration functionality to preform ECDSA verify and ECDH(E) to stablish the TLS handshake and secure session establishment.

### **Table of Contents**

Pre	erequisites	1						
Intr	roduction	1						
lco	Icon Key Identifiers							
1	1 Networking Lavers							
2	TIS and Cryptography	1						
2	2.1 Transport Laver Security							
	2.2 Symmetric Encryption	5						
	2.3 Asymmetric Encryption	5						
	2.4 Digital Signatures	6						
3	TLS Record Protocol	7						
4	TLS Handshake Protocol	7						
	4.1 Understanding the TLS Handshake Procedure	8						
	4.1.2 TLS Client Hello	9						
	4.1.3 TLS Server Hello	10						
	4.1.4 ILS Certificate	11 12						
	4.1.5 TLS Server Reguest	12 13						
	4.1.7 TLS Hello Done	13						
	4.1.8 TLS Certificate	15						
	4.1.9 TLS Client Key Exchange	15						
	4.1.10 TLS Certificate Verify	16						
	4.1.11 TLS Client Change Cipher Spec	16						
	4.1.12 TLS Client Finish	17						
	4.1.1.3 ILS Server Change Cipner Spec	. 17						
	4.1.14 TES Server Finish	17						
F		40						
5		10						
	5.1 Public Key Infrastructure	18						
	5.2 Using ECC508A in a Public Key Intrastructure	19						
6	TLS and ECC508	20						
7	Connecting to AWS IoT using ATECC508A	21						
	7.1 Creating the Root of Trust and Production Signer	22						
	7.2 Registering our Production Signer with AWS IoT	23						
	7.2.2 AWS IoT BYOC Hands On	24						
	7.3 Just-In-Time Registration of the AWS IoT device	35						
	7.3.1 AWS loT TLS and JITR Hands- On	35						
8	Appendix	43						
	8.1 Lambda Function, Policy Attachment and Device Activation	43						
9	License Information	47						



### Icon Key Identifiers

<u>\</u> _	TIP	Highlights Useful Tips and Techniques
i	INFO	Highlights Objectives to be Completed
	RESULT	Highlights the Expected Result of an Assignment Step
	WARNING	Indicates Important Information
•	EXECUTE	Highlights Actions to be Executed Out of the Target



### **1** Networking Layers

The internet at its core is built on top protocols called Internet Protocol (IP) and Transmission Control Protocol. These are used to package data into small packages to be transport. Because the core protocol don't provide any security at all by themselves anyone with access to the communication link can gain full access to the data as well as change the traffic without detection.

When encryption is taken un to account, the attacker might be able to gain access to the encrypted data, but it wouldn't be able to decrypt it or modify it. To prevent impersonation attacks TLS, rely on another important technology besides cryptography called *Public Key Infrastructure* **PKI** which ensures that the traffic is sent to the correct recipient.

To have a better understanding where TLS fit, we will look at the Open Systems Interconnection (OSI) model. This is a conceptual model that can be used to explain network communications. All functionality is mapped in 7 different layers. The bottom layer is the closest to the physical communication link and at the top is the application layer.

#	OSI Layer	Description	Example protocols
7	Application	Application data	HTTP, SMTP, IMAP
6	Presentation	Data representation, conversion, encryption	SSL/TLS
5	Session	Management of multiple connections	-
4	Transport	Reliable delivery of packets and streams	TCP, UDP
3	Network	Routing and delivery of datagrams between network nodes	IP, IPSec
2	Data link	Reliable local data connection (LAN)	Ethernet
1	Physical	Direct physical data connection (cables)	CAT5

Figure 1-1. Open Systems Interconnection (OSI) model

TLS sits above the TCP but below the higher-level protocols such as HTTP.

### 2 TLS and Cryptography

Transport Layer Security (TLS) is a protocol that provides communication security to communications on the internet. It is the most widely security protocol used today. As we will see in subsequent sections TLS is composed of two layers: The TLS Handshake Protocol Layer that allows the Server and Client to authenticate each other, and the TLS Record Protocol Layer which provides connection security.

### 2.1 Transport Layer Security

As mentioned earlier, the TLS protocol protects the communication link or transport layer, which is where the name comes from.

Security is not the only goal of TLS. It actually has four main goals:

### • Cryptographic Security

This is the main purpose of TLS, to enable secure communications between any two parties who wish to exchange information.

#### • Interoperability

It should be possible for programs and libraries to be created and are able to communicate with each other using common cryptographic parameters.

#### • Extensibility

TLS is effectively a framework for development and deployment of cryptographic protocol

#### 2.2 Symmetric Encryption

Symmetric encryption is a method for obfuscation that enables secure transport of data over insecure communication channels. This method is also known as private-key cryptography by the fact that it uses the same cryptographic keys for encrypting the plain-text and decryption of the cipher-text.



# Figure 2-1. Bob and Alice share the same Private-Key for encryption and decryption

#### 2.3 Asymmetric Encryption

Symmetric encryption does a great job t handling large amount dog data at great speeds, but it's not that efficient as soon as the number of parties involved increases:

• Members of the same group must share the same key. The more people join in the group, the more expose the group is.

- A different key could be used for each person joining the group, but this collapses as the group gets bigger
- Symmetric encryption can't be used on unattended systems to secure data. This is do the fact that the process can be reversed by using the same key. A compromise to such a system leads to the compromise of all the data stored in the system.

Asymmetric encryption is a different approach to encryption that uses two keys instead of one. One of the keys is called a Private-Key and the other one is known as the Public-Key. As the name indicates one of the keys must be kept private and the other can be shared.

There's a special mathematical relationship between these keys that enables useful features such as sign-verify or encryption-decryption.

Asymmetric encryption makes communication in large groups easier. This is because you can share your public key widely and anyone can send you a message that you can read and also verify by them signing the message with their private key.



Figure 2-2. Bob uses a Public-Key to encrypt and Alice uses a Private-Key to Decrypt

#### 2.4 Digital Signatures

Digital Signatures is a cryptographic scheme that allows us to verify the authenticity of a digital message or document. The Message Authentication Code (MAC) is a type of digital signature. A MAC is a cryptographic function that extend hashing with authentication, in other words, is a keyed-hash. Only does in possession of the hashing key can produce a valid MAC.

Digital signatures are possible with the help of public key cryptography. Its asymmetric nature can be exploit to device an algorithm that allows a message to be signed by a private key and be verified with its corresponding public key. The sign and verify process depends on the selected public key cryptosystem. For example, Elliptic Curve Cryptography (ECC), where the ECDSA cryptographic algorithm will sign a message using a private key and verify the message using its corresponding public key.

The process for signing a message is as follows

- 1. Calculate the hash of the document you want to sign using for example SHA256
- 2. Use ECDSA to sign the hash of the data with the private key
- 3. To verify the message, the public key associated with the private key needs to be send with the message and its signature
- 4. Used ECDSA to verify the message, having as inputs the Hash of the message, its signature and the public key.

### 3 TLS Record Protocol

TLS is a cryptographic protocol designed to secure communications that consist of an arbitrary number of messages between two parties. At a high level, TLS is implemented via the record protocol, which is in charge of the following aspects of the communication:

#### Message Transport

The record protocol transports opaque data buffers submitted to it by other layers in packets of max 16382 bytes. If a data payload is longer, it will split it into smaller chucks.

#### • Encryption and Message Validation

Initially in a brand new connection, messages are transported without any protection. This is necessary so the first negotiation can take place. However, once the handshake is complete, the record layer starts to apply encryption and integrity validation according to the negotiated connection parameters.

#### • Compression

This feature is no longer used.

#### • Extensibility

The record protocol takes care of data transport and encryption, but delegates all other features to sub-protocols. This approach makes TLS extensible, because new sub-protocols can be added easily. With encryption handled by the record protocol, all sub-protocols are automatically protected using the negotiated connection parameters.

### 4 TLS Handshake Protocol

The TLS Handshake is the process during which the sides negotiate connection parameters and perform authentication. There can be many variations in the exchange, depending on the configuration and supported protocol extensions, but three are the ones that are most widely use:

- Full Handshake with server authentication
- Abbreviated handshake that resumes an earlier session
- Handshake with client and server authentication

In this document we will present the Full Handshake with Server Authentication

#### 4.1 Understanding the TLS Handshake Procedure

Every TLS connection starts with a handshake. If the client hasn't previously established a session with the server, the two sides will execute a full handshake in order to negotiate a TLS session. During this handshake, the client and server will perform four main activities:

- Exchange capabilities and agree on desired connection parameters
- Validate the presented certificates or authentication
- Agree on a shared master secret that will be used to protect the session
- Verify that the handshake messages haven't been modified by a third party

In practice, steps 2 and 3 are part of a single step called **key exchange** or **session establishment** 

This is the sequence followed by a complete TLS handshake with server authentication

- 1. Client begins a new handshake and submits its capabilities to the server
- 2. Server selects connections parameters
- 3. Server sends its certificate chain
- 4. Depending on the selected key exchange, the server sends additional information required to generate the master secret
- 5. Server communicates acceptable Certificate Public Key and Signature Algorithm
- 6. Server indicates completion of its side of the negotiation
- 7. Client sends Certificate Chain
- 8. Client sends additional information required to generate the master secret
- Client proves the possession of the Private Key corresponding to the Public Key in previously sent Certificate by signing all the Hand Shake messages exchanged until this point
- 10. Client switches to encryption and informs the Server
- 11. Ready for Encrypted Application Data
- 12. Server switches to encryption and informs the Client
- 13. Ready for Encrypted Application Data



At this point the connection is established and the parties can begin to send application data securely.

Client	:	Server
1	ClientHello	•
	ServerHello	2
	Certificate	3
	ServerKeyExchange	4
	Certificate Request	5
	ServerHelloDone	6
7	Certificate	•
8	ClientKEyExchange	•
9	Certificate Verify	•
10	ChangeCipherSpec	
1	Finish	
	ChangeCipherSpec	12
	- Finished	13

Figure 4-1. Full TLS handshake with both Client and Server Authentication

#### 4.1.2 TLS Client Hello

The Client Hello message is always the first message sent in a new handshake. It's used to communicate the client capabilities and preferences to the server. Clients send this message at the beginning of a new connection. A TLS session empowered by the ATECC508, will have the Client Hello message that looks like the one below.

	•		WireSharkLog_T	raining.pcapng								
	📶 📕 🖉 💿 🛅 🖹 🖉 🔍 🗢 🏓 🖉 🖌 👤 🥃 🗮 🔍 Q. Q. II											
📕 tcp	🛛 tcp.port == 8883 or tcp.port == 55626 or ip.src == 192.168.2.4 🛛 🖉 📑 🕹											
No.	Time	Source	Destination	Protocol	Length		Info					
	23 1.711252	192.168.2.4	54.200.238.99	TCP		54	57326 → 888	3 [ACK]	Seq=1 Ack=1 Win=433	8 Len=0		
	24 1.960907	192.168.2.4	54.200.238.99	TLSV1.2		114	Client Hell		Con-1 Ack-61 Win-17	022   05-0		
► En	25 2.070139 ame 24: 114 bytes on wi	54.200.238.99 ire (912 hits), 114 hv	192.108.2.4 tes cantured (912 hits) on interfa	CE Ø		54	8883 → 5732	DIACK	Seg=1 ACK=61 Win=1/	922 Len=0		
▶ Et	hernet II. Src: Newport	M f4:16:b6 (f8:f0:05:	f4:16:b6). Dst: f6:5c:89:bc:b3:64	(f6:5c:89:bc:b3:64	1)							
▶ In	ternet Protocol Version	4, Src: 192.168.2.4,	Dst: 54.200.238.99		.,							
► Tr	ansmission Control Prot	tocol, Src Port: 57326	(57326), Dst Port: 8883 (8883), S	eq: 1, Ack: 1, Ler	n: 60							
▼ Se	cure Sockets Layer											
•	TLSv1.2 Record Layer:	Handshake Protocol: Cl	ient Hello									
	Content Type: Handsh	nake (22)										
	Version: TLS 1.2 (0)	<0303)										
	Length: 55	C14+ 11-11-										
	Handshake Protocol: Handshake Type: C	lient Hello (1)										
	length: 51	Lient netto (1)										
	Version: TIS 1.2	(0x0303)										
	v Random	(0,0505)										
	GMT Unix Time:	Aug 18, 2026 07:33:41	.000000000 MDT									
	Random Bytes: 2	2fe16f0b87127aace7259e	7c4a43c06218e4876084413284									
	Session ID Length	: 0										
	Cipher Suites Len	gth: 2										
	▼ Cipher Suites (1	suite)										
	Cipher Suite:	TLS_ECDHE_ECDSA_WITH_A	ES_128_GCM_SHA256 (0xc02b)									
	Compression Metho	ds Length: 1	Cipher Suite supported by Client									
	<ul> <li>Compression Metho</li> <li>Compression Metho</li> </ul>	ds (1 method)										
	Extensions Length	• 8										
	Extension: signat	ure algorithms										
	Type: signature	e algorithms (0x000d)										
	Length: 4											
	Signature Hash	Algorithms Length: 2										
	▼ Signature Hash	Algorithms (1 algorit	hm)									
1	▼ Signature Ha	sh Algorithm: 0x0403										
	Signature	Hash Algorithm Hash:	SHA256 (4)									
	Signature	Hash Algorithm Signat	ure: ECDSA (3)									
0 7	Cipher Suite (ssl.handshake.	ciphersuite), 2 bytes				Packet	ts: 92 · Displayed	92 (100.0%	6)	Profile: Default		

Figure 4-2. ECC508A TLS Client Hello Message

In this case the client sends to the server that its capabilities are:

- Elliptic Curve secp256r1
- ECDHE
- ECDSA
- AES128
- SHA256

Based on ECC508A capabilities (ECC p256, ECDHE, ECDSA and SHA256)

#### 4.1.3 TLS Server Hello

The TLS Server Hello message will communicate the selected connection parameters back to the client. This message is similar in structure to Client Hello but contains only one option per field. A TLS session empowered by the ATECC508, will have the Server Hello message that looks like the one below.

	•			🔚 WireSharkLog_	g_Training.pcapng
	<b>I</b> 🖉 🕑 🔳	े 🛛 🖸 🤇	🔶 🔿 警 🚡 🛓 📃 🤆	Q Q II	
📕 tcp	port == 8883 or tcp.port ==	= 55626 or ip.src == 192	2.168.2.4		Expression +
No.	Time	Source	Destination	Protocol	Length Info
	25 2.070139	54.200.238.99	192.168.2.4	TCP	54 8883 → 57326 [ACK] Seq=1 Ack=61 Win=17922 Len=0
1	26 2.072456	54.200.238.99	192.168.2.4	TCP	1442 [TCP segment of a reassembled PDU]
+	27 2.073982	54.200.238.99	192.168.2.4	TLSv1.2	984 Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
	28 2.083061	192.168.2.4	54.200.238.99	TCP	54 57326 → 8883 [ACK] Seq=61 Ack=1389 Win=4338 Len=0
	29 2.083450	192.168.2.4	54.200.238.99	TCP	54 57326 → 8883 [ACK] Seq=61 Ack=2319 Win=4338 Len=0
<ul> <li>Fra</li> <li>Ett</li> <li>In*</li> <li>Tra</li> </ul>	ame 27: 984 bytes on w hernet II, Src: f6:5c: ternet Protocol Versio ansmission Control Pro	ire (7872 bits), 9 89:bc:b3:64 (f6:5c n 4, Src: 54.200.2 tocol, Src Port: 8	84 bytes captured (7872 bits) on interfa :89:bc:b3:64), Dst: NewportM_f4:16:b6 (f 38.99, Dst: 192.168.2.4 883 (8883), Dst Port: 57326 (57326), Seq	ce 0 8:f0:05:f4:16:b6] : 1389, Ack: 61,	6) , Len: 930
▶ [2	Reassembled TCP Segme	nts (2318 bytes):	#26(1388), #27(930)]		
▼ Se	cure Sockets Layer				
•	TLSv1.2 Record Layer: Content Type: Hands Version: TLS 1.2 (@ Length: 2313	Handshake Protocol hake (22) ix0303)	: Multiple Handshake Messages		
	Handshake Protocol:	Server Hello			
	Handshake Type:	Server Hello (2)			
	Version: TLS 1.2	(0×0303)			
	▼ Random				
	GMT Unix Time:	Sep 1, 2016 11:0	9:50.000000000 MDT		
	Random Bytes:	6190fdda767acb660c	9926aa225a308678f91cf73462ee5f		
	Session ID Lengt	h: 32			
	Session ID: 57c8 Cipher Suite: TL Compression Meth	60de0c5cef604e1bfe4 S_ECDHE_ECDSA_WITH_ od: null (0)	16041aab1887bbc9c1278e186f AES_128_GCM_SHA256 (0xc02b) Ciph	er Suite confi	firmed to Client

Figure 4-3. ECC508A TLS Server Hello Message

Now, the Server will acknowledge the same connection parameters that the Client requested

- Elliptic Curve secp256r1
- ECDHE
- ECDSA
- AES128
- SHA256

Based on ECC508A capabilities (ECC p256, ECDHE, ECDSA and SHA256)

#### 4.1.4 TLS Certificate

In the TLS Certificate message, the Server will carry its X.509 certificate chain. These are provided one after another in **ASN.1 DER encoding**. The main certificate must be sent first, with all of the intermediate certificates following in the correct order. A TLS session empowered by the ATECC508, will have the Certificate message that looks look like the one below.

•		WireSharkLog	Training.ocaong	
			annig peoping	
	N 🖌 🔶 🛸 🐺 🔽 💶 I	$\equiv$ $\alpha \alpha \alpha \pi$		
.port == 8883 or tcp.port == 55626 c	rip.src == 192.168.2.4			Expression +
Time Source	Destination	Protocol	Length Info	
25 2.070139 54.20	a.238.99 192.168.2.4	TCP	54 8883 → 57326 [ACK] Seq=	=1 Ack=61 Win=17922 Len=0
26 2.072456 54.20	a.238.99 192.168.2.4	TCP	1442 [TCP segment of a reass	sembled PDUJ
27 2.073982 54.20	8.238.99 192.168.2.4	TLSV1.2	984 Server Hello, Certifica	ate, Server Key Exchange, Certificate Request, Server Hello Done
20 2.003001 192.1	68 2 4 54 200 238 00	TCP	54 57326 → 8883 [ACK] Seg-	-61 Ack-2319 Win-4338 Len-0
25 21005450 1521		101	54 57520 4 6665 [Held Sed-	-01 ACK-2010 W11-4000 ECH-0
Compression Method: pul	ECDSA_WITH_AES_128_GCM_SHA256 (0xc02D)			
Handshake Protocol: Certif	cate			
Handshake Type: Certifi	ate (11)			
Length: 2049				
Certificates Length: 20/	6			
▼ Certificates (2046 byte:	)			
Certificate Length: 9	16			
v Certificate: 30820386	3082032da0030201020210148b52d5d212a92bb7	. (id-at-commonName=*.iot.	us-west-2.amazonaws.com,id-at-org	ganizationName=Amazon.com, Inc.,id-at-localityName=Seattle,id-at-sta
<pre>v signedCertificate</pre>				
version: v3 (2)	148552d5d212a02bb70b8dead0278c18			
w signature (iso.?	.840.10045.4.3.2)			
Algorithm Id:	1.2.840.10045.4.3.2 (iso.2.840.10045.4.3.	2)		
▼ issuer: rdnSeque	nce (0)			
rdnSequence:	items (id-at-commonName=Symantec Class 3	ECC 256 bit SSL CA - G2,i	d-at-organizationalUnitName=Syman'	tec Trust Network,id-at-organizationName=Symantec Corporation,id-at
▶ validity				
v subject: rdnSequ	ence (0)			
▶ rdnSequence:	<pre>items (id-at-commonName=*.iot.us-west-2.</pre>	amazonaws.com,id-at-organi	zationName=Amazon.com, Inc.,id-at-	-localityName=Seattle,id-at-stateOrProvinceName=Washington,id-at-co
<ul> <li>subjectPublickey</li> </ul>	INTO			
+ algorithm (10-	d: 1.2.840.10045.2.1 (id=ecPublicKey)	Cruptograpic Alg	withm for Dublic Kowic Elli	intic Curve coop3E6x1
v ECParameter	s: namedCurve (0)	Cryptograpic Aigo	intrini for Public Key is Elli	pric curve secp25011
namedCur	/e: 1.2.840.10045.3.1.7 (secp256r1)			
Padding: 0		-		
subjectPublic	ey: 0444baadda02f4621ad86d0c4e3838225a6a1	173df2a602a		
▶ extensions: 8 it	ems			
algorithmIdentifie	(iso.2.840.10045.4.3.2)			
Padding: 0				
Contificate Length: 1	40/5910004/00510941142080400800055040981.			
v Certificate: 3082046c	30820352a00302010202103f9287be9d1da4a37a	. (id-at-commonName=Symant	ec Class 3 ECC 256 bit SSL CA - G	2.id-at-organizationalUnitName=Symantec Trust Network.id-at-organiz
<pre>v signedCertificate</pre>				-,,,
version: v3 (2)				
serialNumber: 0x	3f9287be9d1da4a37a9df6282e775ac4			
⊤ signature (sha25	5WithRSAEncryption)			
Algorithm Id:	1.2.840.113549.1.1.11 (sha256WithRSAEncry	ption)		
v issuer: rdnSeque	nce (0)			
▶ rdnSequence:	items (id-at-commonName=VeriSign Class 3	Public Primary Certificat	ion ,id-at-organizationalUnitName=	=(c) 2006 VeriSign, Inc For auth,id-at-organizationalUnitName=Ve
▶ validity	(0)			
subject: ranseque subjectPublicKet	Info			
subjectrubtickey     id     id	ecPublicKev)			
Padding: 0				
subjectPublic	ev: 040f1ba491d7e7ace7d14e4eb7645be18f7f6	e04d3ab38db		
▶ extensions: 8 it	ams			
	(cha256WithRSAEncryption)			
▶ algorithmIdentifie	(and 200ml childsheller yperon)			
▷ algorithmIdentifie Padding: 0	(Shu250W1Christeneryperon)			

Figure 4-4. ECC508A TLS Certificate Message

### 4.1.5 TLS Server Key Exchange

The purpose of the Server Key Exchange message is to carry additional information that is needed for the key exchange. A TLS session empowered by the ATECC508, will have the Server Key Exchange message that looks like the one below.



				H WireSharkLog_Tr	ining.pcapng				
4 🔳	📶 📕 🧶 🐵 🚞 🛅 🖄 🙆 🗣 🌧 響 🖌 🛬 🚍 🔍 🔍 🍳 🍳 🔍 🗄								
tcp.port =	= 8883 or tcp.port == !	55626 or ip.src == 192.168.2.4					Expression +		
No.	Time	Source	Destination	Protocol Le	ngth Info				
	25 2.070139	54.200.238.99	192.168.2.4	TCP	54 8883 → 57326 [AC	K] Seq=1 Ack=61 Win=17922 Len=0			
+	26 2.072456	54.200.238.99	192.168.2.4	TCP	1442 [TCP segment of a	a reassembled PDU]			
+	27 2.073982	54.200.238.99	192.168.2.4	TLSv1.2	984 Server Hello, Ce	rtificate, Server Key Exchange, Certificate Req	uest, Server Hello Done		
	28 2.083061	192.168.2.4	54.200.238.99	тср	54 57326 → 8883 [AC	KJ Seq=61 Ack=1389 Win=4338 Len=0			
	29 2.083450	192.168.2.4	54.200.238.99	TCP	54 57326 - 8883 LACI	KJ Seq=61 Ack=2319 Win=4338 Len=0			
▶ Frame 2:	7: 984 bytes on wi	re (7872 bits), 984 bytes captur	red (7872 bits) on interface	0					
▶ Etherne	II, Src: 16:5c:8	9:bc:b3:64 (16:5c:89:bc:b3:64),	Dst: NewportM_14:16:66 (18:	10:05:14:16:66)					
▶ Internet	Protocol Version	4, Src: 54.200.238.99, Dst: 192	2.168.2.4						
▶ Transmi	ssion Control Prot	ocol, Src Port: 8883 (8883), Ds1	t Port: 57326 (57326), Seq: 1	1389, ACK: 61, L	n: 930				
▶ [2 Reas	sembled ICP Segmen	ts (2318 bytes): #26(1388), #2/0	(930)]						
+ Secure :	2 Record Lavers k	Jandshake Brotosal: Multiple Han	debako Moseagos						
* 12341	tent Type: Handsh	ake (22)	ushake nessayes						
Vo	reion: TIS 1 2 (Av	0202)							
ve Lei	oth: 2313								
w Ha	dshake Protocol:	Server Hello							
	Handshake Type: Se	erver Hello (2)							
	Length: 70								
	Version: TLS 1.2	(0x0303)							
<b>.</b> .	Random								
	GMT Unix Time:	Sep 1, 2016 11:09:50.000000000	MDT						
	Random Bytes: 6	190fdda767acb660c9926aa225a3086	78f91cf73462ee5f						
	Session ID Length:	: 32							
	Session ID: 57c86	0de0c5cef604e1bfe46041aabf887bbc	9cf278e186f						
	Cipher Suite: TLS	_ECDHE_ECDSA_WITH_AES_128_GCM_SH	IA256 (0xc02b)						
	Compression Method	d: null (0)							
⊳ Ha	ndshake Protocol:	Certificate							
▼ Ha	ndshake Protocol:	Server Key Exchange							
	Handshake Type: Se	erver Key Exchange (12)							
	Length: 144								
•	EC Diffie-Hellman	Server Params							
	Curve Type: nam	ied_curve (0x03)							
	Nameu Curve: se	ccp25011 (0X001/)							
	Pubkey Length:	00 0dd2a654202ba0010bboaac1560-362	deheb14022	Dublic Koute	he exchange for EC	DUE			
	- Signature Hach	Algorithm: 0v0403	Jervers	rubic Key tt	be exchange for EC	DHL			
	Signature Has	sh Algorithm Hash: SHA256 (4)							
	Signature Ha	sh Algorithm Signature: FCDSA (3	n -						
	Signature Lengt	h: 71							
	Signature: 3045	022100f123a3b5cf71f57b93c5bf1f2	5a922907ddc63						

Figure 4-5. ECC508A TLS Server Key Exchange Message

In this case, the server will exchange its Public-Key with the Client. This will be used during the creation or the encryption key by applying ECDHE. This process takes place after the certificate chain has been verified.

#### 4.1.6 TLS Certificate Request

By sending the TLS Certificate Request the Server request the client authentication and communicates acceptable Certificate, Public-Key and Signature algorithms to the Client. A TLS session empowered by the ATECC508, will have the Server Certificate Request message that looks like the one below

	i 🖉 💿 🖿 🚺	े 🎗 🙆 🤇 🔶 🖻	T 🛓 📜	0.0.0.1				
tcp.p	ort == 8883 or tcp.port ==	55626 or ip.src == 192.168.2.4				Expression.	. +	
No.	Time	Source	Destination	Protocol Ler	ath Info			
	25 2.070139	54.200.238.99	192.168.2.4	TCP	54 8883 → 57326	6 [ACK] Seq=1 Ack=61 Win=17922 Len=0		
+	26 2.072456	54.200.238.99	192.168.2.4	TCP	1442 [TCP segment	t of a reassembled PDU]		
	27 2.073982	54.200.238.99	192.168.2.4	TLSv1.2	984 Server Hello	o, Certificate, Server Key Exchange, Certificate Request, Server Hello I	Done	
	28 2.083061	192.168.2.4	54.200.238.99	TCP	54 57326 → 8883	3 [ACK] Seq=61 Ack=1389 Win=4338 Len=0		
	29 2.083450	192.168.2.4	54.200.238.99	TCP	54 57326 → 8883	3 [ACK] Seq=61 Ack=2319 Win=4338 Len=0		
v	<ul> <li>Handshake Protocol: Handshake Type: S Length: 70</li> <li>Version: TLS 1.2</li> <li>Random GMT Unix Time: Random Bytes: Session ID Length</li> <li>Session ID: 57c86</li> <li>Cipher Suite: TLS</li> </ul>	Server Hello Server Hello (2) (0x0303) Sep 1, 2016 11:09:50.00000000 5109fdda767acb660c9926aa225a388 : 32 SudeeSccr6de41bfe46041aabf887bb Sucche EcDSA_WITH_AES_128_GCM_5	) MDT 778f91cf73462ee5f c9cf278e186f HA256 (0xc02b)					
	Compression Metho	Contificate						
	Handshake Protocol:	Certificate						
	Handshake Protocol:	Certificate Request						
	Handshake Type: 0	Certificate Request (13)						
	Length: 30							
	Certificate types	s count: 3						
	▶ Certificate types	s (3 types)						
	Signature Hash Al	Igorithms Length: 22						
		lgorithms (11 algorithms)						
	▼ Signature Hash	Algorithm: 0x0603						
	Signature Ha	ash Algorithm Hash: SHA512 (6)						
	Signature Ha	ash Algorithm Signature: ECDSA (	3)					
	▼ Signature Hash	Algorithm: 0x0601						
	Signature Ha	ash Algorithm Hash: SHA512 (6)						
	Signature Ha	ash Algorithm Signature: RSA (1)						
	▼ Signature Hash	Algorithm: 0x0503						
	Signature Ha	ash Algorithm Hash: SHA384 (5)						
	Signature Ha	ash Algorithm Signature: ECDSA (	3)					
	▼ Signature Hash	Algorithm: 0x0501						
	Signature Ha	ash Algorithm Hash: SHA384 (5)						
	Signature Ha	ash Algorithm Signature: RSA (1)						
	Signature Hash	Algorithm: 0x0403	Hashing an	d Signature algori	thm			
	Signature Ha	ash Algorithm Hash: SHA256 (4)		hutha Comucita C	1			
	Signature Ha	ash Algorithm Signature: ECDSA (	3) requested	by the Server to C	ient			
	▼ Signature Hash	Algorithm: 0x0401						
	Signature Ha	ash Algorithm Hash: SHA256 (4)						
	Signature Ha	ash Algorithm Signature: RSA (1)						
	<ul> <li>Signature Hash</li> </ul>	Algorithm: 0x0303						
	Signature ha	ash Algorithm Hash: SHA224 (3)	2)					
	- Cisseture Hech	Aleesithe, 0.0201	37					
	* Signature Hash	Acgorithm Hach: SHA224 (2)						
	Signature Ha	ash Algorithm Signature: BSA (1)						
	Signature Hash	Algorithm: 0x0203						
1	Signature Ha	ash Algorithm Hash: SHA1 (2)						
1	Signature Ha	ash Algorithm Signature: ECDSA (	3)					
1	Signature Hash	Algorithm: 0x0201						
	Signature Ha	ash Algorithm Hash; SHA1 (2)						
1	Signature Ha	ash Algorithm Signature: RSA (1)						
1	Signature Hash	Algorithm: 0x0202						
1	Signature Ha	ash Algorithm Hash: SHA1 (2)						
1	Signature Ha	ash Algorithm Signature: DSA (2)						
		,						

Figure 4-6. ECC508A TLS Server Certificate Request Message

#### 4.1.7 TLS Hello Done

The Server Hello message indicates that the Server has sent all necessary messages for the Hand Shake to take place. The Server will now wait for further messages from the client.

	•			📑 WireSharkLo	.og_Training.pcapng			
1	<b>1</b> ( )	N 🖸 🤉 🦛	👄 🕾 🛧 👲 📜 📃	0,0,0,1	TT			
I tcp.	port == 8883 or tcp.port ==	55626 or ip.src == 192.168.2.4			Ξ	Expression +		
No.	Time	Source	Destination	Protocol	Length Info			
T	25 2.070139	54.200.238.99	192.168.2.4	TCP	54 8883 → 57326 [ACK] Seq=1 Ack=61 Win=17922 Len=0			
4	26 2.072456	54.200.238.99	192.168.2.4	TCP	1442 [TCP segment of a reassembled PDU]			
+	27 2.073982	54.200.238.99	192.168.2.4	TLSv1.2	984 Server Hello, Certificate, Server Key Exchange, Certificate Request	, Server Hello Done		
	28 2.083061	192.168.2.4	54.200.238.99	TCP	54 57326 - 8883 [ACK] Seq=61 Ack=1389 Win=4338 Len=0			
	29 2.083450	192.168.2.4	54.200.238.99	TCP	54 57326 → 8883 [ACK] Seq=61 Ack=2319 Win=4338 Len=0			
Fra Ett Ett Fra Fra Fra Fra Fra Fra Fra Fra Fra Fra	29 2.08358         192.108.2.4         54.200.23.99         TCP         54 57226 - 8883 [ACK] Seqe61 Acke-2319 Min-4338 Len=0           > Therea 27: 96 bytes on virce (772 bits) on 964 bytes capture (772 bits) on interface 0         >           > Therea T1, Str: (65:509:bc:b3:64 (fi:55:09:bc:b3:64 (fi:56:09:bc:b3:64 (fi:61:00:bc:b3:64 (fi:61:00:b); fi:61:00:b:f:fi:61:00:bc:b3:64 (fi:61:00:b); fi:61:00:b:fi:61:00:b:fi:61:00:fi:61:							
	Handshake Protocol:	Certificate Request						
	Handshake Protocol: Handshake Type: S	Server Hello Done (14)						
	Length: 0	erver netto Done (14)						
	Long citt 0							

Figure 4-7. ECC508A TLS Server Hello Done Message



#### 4.1.8 TLS Certificate

In the TLS Client Certificate Message, Client sends its Certificate chain to Server.



Figure 4-8. ECC508A TLS Client Certificate Message

#### 4.1.9 TLS Client Key Exchange

The Client Key Exchange message carries the Client's contribution to the key exchange. Its content depends on the negotiated cipher suite. A TLS session empowered by the ATECC508 will have the Client Key Exchange message that looks like the one below

000		📓 WireSharkLog_Training.pcapng						
🚄 🔳 🔬 🐵 🖿 🖺 🔀 🔍 🖛 🛛	🔶 🖀 🐔 👱 📃 🗎	R Q Q II						
tcp.port == 8883 or tcp.port == 55626 or ip.src == 192.168.2.4			Expression +					
No. Time Source	Destination	Protocol Length Info						
28 2.083061 192.168.2.4	54.200.238.99	TCP 54 57326 → 8883 [ACK] Seq=61 Ack=1389 Win=4338 Len=0						
29 2.083450 192.168.2.4	54.200.238.99	TCP 54 57326 → 8883 [ACK] Seq=61 Ack=2319 Win=4338 Len=0						
30 3.536522 192.168.2.4	54.200.238.99	TLSv1.2 1007 Certificate						
31 3.726914 54.200.238.99	192.168.2.4	TCP 54 8883 → 57326 [ACK] Seq=2319 Ack=1014 Win=19060 Len=0						
32 4.282974 192.168.2.4	54.200.238.99	TLSv1.2 129 Client Key Exchange						
<ul> <li>Ethernet II, Src: NewportM_f4:16:b6 (f8:f0:85:f4:</li> <li>Internet Protocol Version 4, Src: 192.168.2.4, Ds:</li> <li>Transmission Control Protocol, Src Port: 57326 (5)</li> <li>Secure Sockets Layer</li> </ul>	L6:b6), Dst: f6:5c:89:bc:b3:64 ( t: 54.200.238.99 7326), Dst Port: 8883 (8883), Se	f6:5c:89:bc:b3:64) q: 1014, Ack: 2319, Len: 75						
<pre>b Transmission Control Protocol, Src Port: 57326 (57326), Dst Port: 8883 (8883), Seq: 1014, Ack: 2319, Len: 75 v Secure Sockets Layer T.LSv1.2 Record Layer: Handshake Protocol: Client Key Exchange Content Type: Handshake Protocol: Client Key Exchange Length: 78 Handshake Protocol: Client Key Exchange Handshake Type: Client Key Exchange Length: 66 v EC Diffic-Hellman Client Params Pubkey Length: 65</pre>								

Figure 4-9. ECC508A TLS Client Key Exchange Message

#### 4.1.10 TLS Certificate Verify

The Client uses the Certificate Verify to prove the possession of the Private Key corresponding to the Public Key in the previously sent Client Certificate. This message contains a Signature of all the handshake messages exchanged until this point. A TLS session empowered by the ATECC508 will have the Certificate Verify message that looks like the one below

_							
••	•			🔚 WireSharkLog	_Training.pcapng		
	<b>d</b> 💿 🖿	े 🖹 🙆 🤇	🔶 🍝 🖀 著 🛓 🚊	📃 🔍 Q, Q, 🎹			
📕 tcp.	port == 8883 or tcp.port ==	55626 or ip.src == 19	2.168.2.4				Expression +
No.	Time	Source	Destination	Protocol	Length Info		
	32 4.282974	192.168.2.4	54.200.238.99	TLSv1.2	129 Client Key Exchange	e	
	33 4.422758	54.200.238.99	192.168.2.4	TCP	54 8883 → 57326 [ACK]	Seq=2319 Ack=1089 Win=19060 Len=0	
	34 4.620872	192.168.2.4	54.200.238.99	TLSv1.2	138 Certificate Verify		
	35 4.750975	54.200.238.99	192.168.2.4	TCP	54 8883 → 57326 [ACK]	Seq=2319 Ack=1173 Win=19060 Len=0	
	36 4.791402	192.168.2.4	54.200.238.99	TLSv1.2	60 Change Cipher Spec		
▶ Int ▶ Tra ▼ Sec	ernet Protocol Versio nsmission Control Pro ure Sockets Layer	n 4, Src: 192.168 tocol, Src Port: !	2.4, Dst: 54.200.238.99 ;7326 (57326), Dst Port: 8883 (	8883), Seq: 1089, Ack: 231	9, Len: 84		
v	Content Type: Hands Version: TLS 1.2 (0 Length: 79	Handshake Protoco hake (22) x0303)	L: Certificate Verity				
	<ul> <li>Handshake Protocol: Handshake Type: I Length: 75</li> <li>Signature Hash A Signature Hash Signature Hash Signature Length Signature: 304502</li> </ul>	Certificate Veri Certificate Verify Igorithm: 0x0403 Algorithm Hash: Algorithm Signat 71 2204521f1d2333082a	'y (15) SHA256 (4) Jre: ECDSA (3) 2C078d54d8afa424f9ea20c28	Certificate Chain successfully allov with creation of	has been verified wing for ECDHE to proce encryption key	eed	

Figure 4-10. ECC508A TLS Verify Certificate Message

#### 4.1.11 TLS Client Change Cipher Spec

The Client Change Cipher Spec message is a signal that the Client obtained enough information to create the connection parameters, generated the encryption keys and will now switch to encrypted communications.

![](_page_15_Picture_9.jpeg)

	•			🔓 WireSharkLog_Trair	ing.pcap.ng		
		N 🔊 🙈 O 🚄					
<u> </u>							
📕 top	.port == 8883 or tcp.port ==	55626 or ip.src == 192.168.2	2.4z			Expression H	F.
No.	Time	Source	Destination	Protocol Leng	th Info		
	37 4.903681	54.200.238.99	192.168.2.4	TCP	54 8883 → 57326 [ACK] Seq=2319 Ack=1179 Win=19060 Len=0		
	38 4.927213	192.168.2.4	54.200.238.99	TLSv1.2	99 Hello Request, Hello Request		
	39 5.029493	54.200.238.99	192.168.2.4	TCP	54 8883 → 57326 [ACK] Seq=2319 Ack=1224 Win=19060 Len=0		
	40 5.041669	54.200.238.99	192.168.2.4	TLSv1.2	136 Change Cipher Spec, Hello Request, Hello Request, Encrypted Ale	ert	4
	41 5.041721	54.200.238.99	192.168.2.4	TCP	54 8883 → 57326 [FIN, ACK] Seq=2401 Ack=1224 Win=19060 Len=0		
▶ Fr	ame 40: 136 bytes on w	ire (1088 bits), 136 by	tes captured (1088 bits) on inte	rface 0			
▶ Et	hernet II, Src: f6:5c:	39:bc:b3:64 (f6:5c:89:b	c:b3:64), Dst: NewportM_f4:16:b6	(f8:f0:05:f4:16:b6)			
⊳ In	ternet Protocol Versio	1 4, Src: 54.200.238.99	), Dst: 192.168.2.4				
▶ Tr	ansmission Control Pro	tocol, Src Port: 8883 (	8883), Dst Port: 57326 (57326), 1	5eq: 2319, Ack: 1224, L	en: 82		
▼ Se	cure Sockets Layer						
	TLSv1.2 Record Layer:	Change Cipher Spec Pro	tocol: Change Cipher Spec				
	Content Type: Chang	e Cipher Spec (20)					
	Version: TLS 1.2 (0	<0303)					
	Length: 1						
	Change Cipner Spec	lessage					
	ILSVI.2 Record Layer:	Handshake Protocol: Mu	itiple Handshake Messages				
	Content Type: Hands	1aRe (22)					
	Version: 1L5 1.2 (0.	(0303)					
	= Handshake Protocol	Helle Request					
	Handshake Frotocot:	herto Request (8)					
	Length: 0	letto hequest (0)					
	W Handshake Protocol:	Hello Request					
	Handshake Type: h	lallo Request (A)					
	Length: 0	ice to hequese (o)					
	TLSv1.2 Record Laver:	Encrypted Alert					
1	Content Type: Alert	(21)					
	Version: TLS 1.2 (0	(0303)	ent has all that it needs to	start encrypting	unlication		
	Length: 26		ent has an that it heeus to	start enciypting o	ippication		
	Alert Message: Encr	pted Alert da	ta				

Figure 4-11. ECC508A TLS Client Change Cipher Spec Message

#### 4.1.12 TLS Client Finish

The Client Finish message is a sign that the Handshake process is complete from the Client's perspective

#### 4.1.13 TLS Server Change Cipher Spec

The Server Change Cipher Spec message is a signal that the Server obtained enough information to create the connection parameters, generated the encryption keys and will now switch to encrypted communications.

#### 4.1.14 TLS Server Finish

The Server Finish message is a sign that the Handshake process is complete from the Server's perspective.

#### 4.1.15 TLS Encrypting Message

All application data shared from this point forward will be encrypted data.

### 5 PKI and ECC508

#### 5.1 Public Key Infrastructure

The Public Key Infrastructure (PKI) was created so we can communicate with someone that we have never meet before securely, just by sharing its Public Key. The PKI model relies on a trusted entity known as Certificate Authority which job is to issue trusted Certificates.

This certificate issued by a Certificate Authority will become the systems Root of Trust. The Root of Trust will extend a certificate to the OEM which will turn in to the OEM-CA.

![](_page_17_Picture_4.jpeg)

#### Figure 5-1. Establishing your PKI infrastructure

The OEM-CA will create an intermediate CA or Production Signer. The end device will request a certificate issuance (CSR) to the Signer.

![](_page_17_Figure_7.jpeg)

Figure 5-2. Creating a Production Signer that will sign the End Device CSR's

![](_page_17_Picture_9.jpeg)

The production signer will sign the end device CSR, and thus will give an identity to the end device.

![](_page_18_Picture_1.jpeg)

Figure 5-3. The Production Signer will now Sign the CSR's

#### 5.2 Using ECC508A in a Public Key Infrastructure

The ECC508A is a device that can store Private Keys securely and use these private Keys internally in cryptographic algorithms to support a PKI. The cryptographic algorithms that ECC508A supports are the Elliptic Curve p256, Elliptic Curve Digital Signature Algorithm (ECDSA) and the Elliptic Curve Diffie-Hellman algorithm (ECDH(E)) and SHA256.

- ECDSA
  - It's a signing algorithm that uses elliptic curve cryptography.
- ECDH
  - It's a Key exchange agreement that allows two parties, each having an elliptic curve public-private key, to establish a common key to secure their communications.
- Elliptic Curve p256 Key Generator
- RNG (FIPS Compliant)
  - o Random Number Generator
- SHA256
  - SHA256 Hashing algorithm engine

The ECC508A will also allow you to store compressed X.509 certificates that the Host can decompress.

![](_page_18_Picture_15.jpeg)

These features allow the ECC508A to be used as a cryptographic coprocessor for stablishing a Public Key Infrastructure and also be used in a TLS session establishment between a Host and a Client.

The PKI in its most complete form using an ECC508A device to store the Private Keys and its relevant compressed certificates would look like this.

![](_page_19_Picture_2.jpeg)

Figure 5-4. A Public Key Infrastructure empowered by the ECC508A

### 6 TLS and ECC508

As mentioned in the previous section the ECC508A will act as a cryptographic coprocessor and HW key storage device. During the provisioning phase of the ECC508A, it will create its Elliptic Curve Private Key and lock the slot where it is stored. Once the ECC508A device is configured and locked the secret keys that were stored in the device will never leave it. All the cryptographic operations where this private key is needed will happen within the device itself.

By following this approach, the level of security that TLS offers increases since the private keys will never be exposed in software.

During the TLS handshake and session establishment process the ECC508A will

- Perform the ECDSA verification of the Certificates being exchanged.
- It will use ECDH to generate a shared key that can be used to established a secure session between the Server and the Client

![](_page_19_Picture_10.jpeg)

![](_page_19_Picture_11.jpeg)

	•		WireSharkl	_og_Training.pcapng					
	i 🙆 🙆 📘	🖹 🖸 🔍 🔶	• 🔿 🖀 🐔 👱 其 📕	🔍 Q, Q, 🎹					
📕 tcp.	port == 8883 or tcp.port == 55	626 or ip.src == 192.168	2.4					×	Expression +
No.	Time	Source	Destination	Protocol	Length	Info			
	23 1.711252	192.168.2.4	54.200.238.99	TCP		54 57326	→ 8883 [ACK]	Seq=1 Ack=1 Win=4	1338 Len=0
	24 1.960907	192.168.2.4	54.200.238.99	TLSv1.2		114 Client	t Hello		
b Ees	25 2.070139	54.200.238.99	192.168.2.4	TCP		54 8883 -	→ 57326 [ACK]	Seg=1 Ack=61 Win=	=17922 Len=0
FT FT	ernet TI Src: NewportM	f4.16.66 (f8.f0.05.	f4.16.b6) Det. f6.5c.89.bc.b3	•64 (f6.5c.89.bc.b3.64	0				
► Int	ernet Protocol Version 4	Src: 192.168.2.4.	Dst: 54.200.238.99	104 (10150105100105104	· ·				
► Tra	insmission Control Protoc	ol, Src Port: 57326	(57326), Dst Port: 8883 (8883	), Seg: 1, Ack: 1, Len	: 60				
▼ See	ure Sockets Layer								
	TLSv1.2 Record Layer: Har	ndshake Protocol: Cl	lient Hello						
	Content Type: Handshak	(22)							
	Version: TLS 1.2 (0x03	103)							
	Length: 55								
	Handshake Protocol: Cl	ient Hello							
	Handshake Type: Clie	ent Hello (1)							
	Version: TIS 1.2 (0)	~0202)							
	* Random	x0303)							
	GMT Unix Time: Au	ug 18, 2026 07:33:41	.000000000 MDT						
	Random Bytes: 2fe	16f0b87127aace7259e	7c4a43c06218e4876084413284						
	Session ID Length: (	0							
	Cipher Suites Length	h: 2							
	Tipher Suites (1 suites)	ite)							
	Cipher Suite: TLS	ECDHE_ECDSA_WITH_A	ES_128_GCM_SHA256 (0xc02b)						
	Compression Methods	Length: 1	Cipher Suite supported by Client						
	Compression Methods	(1 method)							
	Compression Metho	)a: null (0)							
	Extensions Length: a	o plansithms							
	Type: signature a	algorithms (0x000d)							
	Length: 4	regoriennis (oxooou)							
	Signature Hash Al	laorithms Lenath: 2							
	<ul> <li>Signature Hash Al</li> </ul>	lgorithms (1 algorit	hm)						
	Signature Hash	Algorithm: 0x0403							
	Signature Ha	ash Algorithm Hash:	SHA256 (4)						
	Signature Ha	ash Algorithm Signat	ure: ECDSA (3)						
0 7	Cipher Suite (ssl.handshake.ciph	hersuite), 2 bytes				Packets: 92 · D	isplayed: 92 (100.0	%)	Profile: Default

Figure 6-1. Using ECC508A ECDSA and ECDH to establish a TLS session.

For a detailed process on how the TLS handshake and session establishment works, please refer to section 4 of this document.

### 7 Connecting to AWS IoT using ATECC508A

Now that we understand how the ATECC508A can be used to harden the TLS session establishment during the handshake and secure public key exchange, we will use it to connect the device to AWS IoT securely. The steps that we will follow basically are:

- Creating the Root CA that will sign our production Signer using our Root Module.
- Register our Signer Certificate with AWS Server by using the Bring Your Own Certificate (BYOC) capability of AWS IoT.
- Establish a secure TLS session using ECC508A to register our IoT device by using the Just In Time Registration capability (JITR) of AWS IoT.
- Securely exchange MQTT messages between our IoT device and AWS IoT once the TLS session has been created.

We have created a GitHub repository containing the necessary information on how to run this hands-on. It has also links to AWS documentation that will support this application note if more detailed information is needed

MicrochipTech Secure Insight on Things GitHub repository

https://github.com/MicrochipTech/AWS-Secure-Insight

# Atmel

#### 7.1 Creating the Root of Trust and Production Signer

From section 5, we know that:

- The purpose of a **Root of CA** is to give authority to the Production Signer to sign the loT device.
  - The Root CA is basically signing the Production Signer's certificate using its Private Key.
  - It will allow the user to stablish another layer of security to its network by securing the application layer and be able to verify the PKI chain back to it.

In our AWS Zero Touch Provisioning kit you will find a Root Module that we have provided for the purpose of creating a Root CA.

![](_page_21_Picture_6.jpeg)

Figure 7-1. Root Module. Contains an ATECC508 configured to be used as a Root CA

- The purpose of the **Production Signer** is to validate the identity of the IoT device.
  - The Identity of the IoT device will be given when the Production Signer signs the End Node's certificate using its Private Key

In our AWS Zero Touch Provisioning kit you will find a Signer Module that we have provided for the purpose of creating the Production Signer.

![](_page_21_Figure_11.jpeg)

Figure 7-2. Signer Module. Contains an ATECC508 configured to be a Signer CA

The creation of the Root CA and the Production Signer and will be done by the GUI in interaction with the Root and Signer Modules. It is the first step that the GUI takes during the process of registering the Production Signer into AWS IoT. Let's review this process in the next section.

![](_page_21_Picture_15.jpeg)

#### 7.2 Registering our Production Signer with AWS IoT

As we have mentioned in the previous sections, our objective is to have IoT devices (end nodes), that will have an identity that AWS IoT can trust. The way we achieve this is by registering our Production Signer CA certificate with AWS IoT along with a verification certificate which will provide the means to verify that you have access to both the Production Signer CA and the private Key associated to it.

To prove you have ownership of the Signer CA private key we generate a verification certificate using the Signer CA certificate, the verification code used to generate a verification certificate and sign it with the Signer CA private key.

Below is the sequence diagram that we will follow to register or Production Signer CA certificate in AWS IoT.

![](_page_22_Figure_4.jpeg)

Figure 7-3. AWS IoT BYOC Registration process using the Insight GUI, Root and Signer modules

#### 7.2.2 AWS IoT BYOC Hands On

The objective of this hands on will be to demonstrate the process that involves registering your Production Signer to AWS IoT by using the BYOC (Bring Your Own Certificate), functionality.

We will be using the following setup:

- AWS IoT device compose of:
  - o ATSAMG55
  - Winc1500 (updated to FW 19.4.4)
  - CryptoAuthXplained Pro
- Secure Insight of Things GUI
- Root Module
- Signer Module

![](_page_23_Picture_10.jpeg)

#### Setting-up your ATSAMG55 AWS IoT device

Our first task is to set up our G55 AWS IoT device. Take out from your **AWS Zero Touch Secure Provisioning kit** the following tools:

- AWS IoT Thing compose of:
  - ATSAMG55
  - Winc1500 (updated to FW 19.4.4)
  - CryptoAuthXplained Pro

Now assemble them as shown in the image below.

- Winc1500 will connect to EXT 1
- CryptoAuthXpro will connect to EXT 4
- OLED1 Explained Pro will connect to EXT3

![](_page_23_Picture_22.jpeg)

![](_page_24_Figure_0.jpeg)

Figure 7-4. ATSAMG55 AWS IoT device

The **AWS Zero Touch Secure Development Kit** comes with a micro USB cable. Connect one end of the USB cable to the **USB Target** connector in ATSAMG55 setup, as shown in Figure 7-5.

![](_page_24_Picture_3.jpeg)

Figure 7-5. Connecting the USB cable to your ATSAMG55 setup.

![](_page_25_Picture_0.jpeg)

#### Installing the AWS Secure Insight Application GUI

We will now download and install the **Installing the AWS Secure Insight on Things Application GUI** from GitHub. Open the link bellow.

https://github.com/MicrochipTech/AWS-Secure-Insight/blob/master/sw/GUI/Windows-x64/AwsSecureIoT.msi

Press the download button.

	nipTech / AWS-	<b>⊙</b> Unwatch	• ●	\star Star	1	<b>∛</b> Fork	2				
<> Code	() Issues 1	위 Pull requests 0	Projects 0	🗏 Wiki	Pulse	III Graphs	🔅 Set	tings			
Branch: ma	ster - AWS-Se	cure-Insight / sw / (	GUI / Windows-x64	4 / AwsSec	ureloT.msi			Fin	d file	Сору ра	ath
🕂 oscara	tmel Updating GUI 1	to new release						835	ie9a3 2	29 days a	igo
1 contribute	or										
58.4 MB							Dov	nload H	istory	Ţ	Ĩ

#### Figure 7-6. Downloading the AWS Secure Insight on Things Application GUI

Open the Installer and run. Follow the instructions from the wizard.

![](_page_25_Picture_8.jpeg)

Figure 7-7. Installing the AWS Secure Insight on Things Application GUI

![](_page_25_Picture_10.jpeg)

After the installation is completed you should see an icon in your desktop like the one bellow.

![](_page_26_Picture_1.jpeg)

Figure 7-8. AWS Secure Insight GUI desktop shortcut

## Connecting the Root and Signer Module and ATSAMG55

The **AWS Zero Touch Secure Development Kit** comes with a USB dongle equipped with an ECC508 configured as a Root Certificate Authority (Red Label) and Signer USB dongle (Green Label).

- Connect the two USB modules to your desktop.
- Connect the ATSAMG55 AWS IoT device to your desktop

#### Configuring the Secure Insight on Things GUI

Now that we have the USB Root and Signer Modules and the ATSAMG55 AWS IoT device connected to our desktop we will run the **AWS Secure Insight** GUI.

The first time you run the AWS Secure Insight GUI, a configuration window will appear like the one shown below in Figure 7-9.

Secure Insight on Things Microchip Insight on Things View Thing Help	-	×
Secure Insight on Things AWS Setup		
AWS Thing Name 1		 
AWS Region Name (example: us-west-2) 2		
AWS Access Key ID 3		
AWS Secret Access Key 4		
WIFI SSID 5		
WiFi Password 6		
CREATE NEW THING 7		
Copyright © Microchip Technology, Inc. Microchip Secure Insight /	Applicatio	

Figure 7-9. Secure Insight on Things configuration setup

The information that we need to enter in the configuration window is as follows

- 1. The name of you AWS IoT device
- 2. The AWS IoT Region Name we will be connecting to. Usually us-west-2
- 3. The AWS IoT Access Key ID. This is provided when you create an AWS IoT account.
- 4. AWS Secret Access Key. This is provided when create an AWS IoT account.
- 5. Your WiFi access point SSID
- 6. Your WiFi access point Password
- 7. After entering all the required information press the **Create New Thing Button**

### Registering your Production Signer

We will now register our Production Signer using the Secure Insight on Things GUI.

Before we proceed we need to make sure that all of our devices (ATSAMG55, Root Module, Signer Module), are being detected. If they are, you should see them enumerated with in the GUI as shown below in Figure 7-10.

Secure Insight on Things
Microchip Insight on Things View Thing Help
Secure Insight on Things Setup
REGISTER SIGNER
PREPARE AWS THING
RE-SCAN CONNECTIONS
AWS Root AT88CKECCROOT
AWS Signer AT88CKECCSIGNER
AWS Thing ATSAMG55 (AWS v1.0.6)
Copyright © Microchip Technology, Inc. Microchip Secu Insight Application v1.0.0

Figure 7-10. ATSAMG55, Root Module and Signer Module enumerated correctly

If you have all of the devices connected to your desktop and still do not see all of them showing up, go to **View** menu and select **Reload**.

# **Atmel**

In order to see the log of the Production Signer registration into AWS IoT, we will use as aid the console view with in our GUI. To enable this view, follow the next steps:

- Open the View menu.
- Select Toggle DevTools from the drop down menu.

Secure Insight on Things	_		Х
Microchip Insight on Things View Thing Help			
Secure Insight (			
Toggle DevTools			
REGISTER SIGNER			
PREPARE AWS THING			
RE-SCAN CONNECTIONS			
AWS Root AT88CKECCROOT			
AWS Signer AT88CKECCSIGNER			
AWS Thing ATSAMG55 (AWS v1.0.6)			
Copyright © Microchip Technology, Inc. Microchip Secure Insight A	pplication	n v1.0.0	)

Figure 7-11. Enabling the Console view of the Secure Insight on Things GUI

![](_page_29_Picture_5.jpeg)

Figure 7-12. Console will show the ongoing process

From the new window pane that opened, select the Console option.

![](_page_29_Picture_8.jpeg)

Before we start our Production Signer and Verification certificate registration we need to make sure that the initialization of the GUI was successful. We will know this because we would have received the AWS IoT device EndPoint which will be used during our process.

		-		Х		
🔍 🛛 Elements Network Sources Timeline Con	sole »	>_	* =	×		
🛇 🗑 <top frame=""> 🔻 🗌 Preserve log</top>						
Found Hid Devices [object Object],[object Object [object Object],[object Object],[object Object]	],	ins:	ight.js:	525		
Microchip Kit Detected: \\? \hid#vid_04d8&pid_0f31#7&3025a1e4&0&0000#{4d1e55 001111000030}	b2-f16f-:	<u>ins:</u> 11cf-8	<u>ight.js:</u> 8cb-	535		
signerSend: b:f(00)		insi	ght.js:1	065		
Microchip Kit Detected: \\? \hid#vid_04d8&pid_0f30#7&1dda532f&0&000#{4d1e55 001111000030}	b2-f16f-:	<u>ins:</u> 11cf-8	<u>ight.js:</u> 18cb-	535		
rootSend: b:f(00)		<u>ins</u> :	ight.js:	905		
signerState: KitName		insight.js:1113				
signerReply: AT88CKECCSIGNER 00(020008)		insight.js:1114				
rootState: KitName		ins:	ight.js:	953		
rootReply: AT88CKECCROOT 00(020008)		<u>ins</u> :	ight.js:	954		
comName: COM6		ins:	ight.js:	608		
pnpId: USB\VID_03EB&PID_2404\6&32A5DFEC&0&1		ins:	ight.js:	609		
manufacturer: ATMEL, Inc.		ins:	ight.js:	610		
comName: COM12		ins:	ight.js:	608		
pnpId: USB\VID_03EB&PID_2111&MI_01\6&31599B4C&08	0001	ins:	ight.js:	609		
manufacturer: Atmel Corp.		ins:	ight.js:	610		
message written		ins:	ight.js:	638		
thingState: KitName		ins:	ight.js:	777		
thingReply: ATSAMG55 00(010006)		ins:	ight.js:	778		
Thing Endpoint: a3adakhi3icyv9.iot.us-west- 2.amazonaws.com		ins:	ight.js:	167		
>						

Figure 7-13. Secure Insight on Things GUI successfully initialized.

The Next steps will take place during the registration process of our Production Signer in to AWS IoT. To Start the process, press the "**Register Signer**" Button.

The following steps will take place during the registration processes as illustrated in the sequence diagram in Figure 7-3:

- 1. Secure Insight GUI will request the Root CA's public key to Root Module.
- 2. Secure Insight GUI will request the Root CA's certificate to the Root Module in PEM format.
- 3. Secure Insight GUI will request the Verification Code from AWS IoT.
- 4. Secure Insight GUI will request the Signer Module to build Verification Certificate.
- 5. **Signer Module** will self-sign the **Verification Certificate**.
- 6. Secure Insight GUI will request Signer Module to build the Production Signer Certificate.
- 7. Secure Insight GUI will request Root Module to sign the Production Signer Certificate.
- 8. Register **Production Signer** and **Verification Certificates** with AWS IoT in PEM format.

<b>Q</b>	Network Sources Timeline Profiles Resources Audits Console	>_  <b>‡ ⊑</b>  ×
🛇 🗑 <top frame=""></top>	▼	
rootState: PubKey		insight.js:953
rootReply: 00(DBF312BECF35DE E7990D728589AE55D0	<b>1</b> 36D2626273125918DEB1DE902FD3EE3EB1061EA74A0772D79CC97D37BA6A82CAEED7A21EE5AC8EI 06)	<u>insight.js:954</u> 5540909B687FA610C
rootSend: aws:g(00	0,00)	insight.js:905
rootState: GetRoot	tCert	insight.js:953
rootReply: 00(032D2D2D2D2D2D2 76A3269593833355A 25842735A534250630 1404468424946A4761 50A4E546C614D4563	4547494E2043455254494649434154452D2D2D2D2D2D0A4D49494238444343415A65674177494241 30677A4469345477457743675949486F5A497A6A304541774977517A45640A4D42734741315545 6D6468626D6C365958527062323478496A416742674E5642414D4D475556345957317762475567 62335167513045774942634E4D5459774E7A45354D6A41774D444177576867504F5468354F5445 784854416242674E5642416F4D46455634)	insight.js:954 674952524C6145657 43677755525868686 0A5156524651304D3 794D7A45794D7A553
rootSend: aws:g(00	0,01)	insight.js:905
rootState: GetRoot	tCert	insight.js:953
rootReply: 00(13595731776247? 2554E444E54413451? 841304941424E767A4 3797537580A6F6837 1494D4159424166384	556754334A6E59573570656D4630615739754D5359774A41594456515144444231460A65474674 534254615764755A584967516846474D7A425A4D424D4742797147534D34394167454743437147 457237504E643432306D4A6963784A5A474E3678337041763028342B7351596570306F48637465 6C72493731514A436261482B60454D35356B4E636F574A726C5851616A5A6A426B4D4249474131 43415141770A4467594456523050415148)	<u>insight.is:954</u> 6347786C494546555 534D34390A4177454 6379583033756D714 5564457745422F775
rootSend: aws:g(00	0,02)	insight.js:905
rootState: GetRoot	tCert	insight.js:953
rootReply: 00(232F4241514441( A42674E56485340454 441674E4841444245( 96747387264376C453 043455254494649434	6748454D423047413155644467515742425174345133484133787A6D426A503736635144485149 4744415767425174345133484133787A6D426A5037366351444851496F47763067444148426767 0A4169423128746C494744574A6436547473417A35783273794C6558682F704844356549286D41 3343735263524773420A4874584145284236696E4C57494573345A4452566E6C4639724E733D0A 4154452D2D2D2D2D0A)	<u>insight.is:954</u> 5F477630674441660 71688B6A4F5051514 544D364B36787A674 2D2D2D2D2D454E442
Root Certificate: BEGIN CERTIF: MIB8BOCAZegAwIBA MBsGA1UECgwURXhhbJ QVRFQ0M1MDhBIFJvb3 NTLaMEcxHTAbBgNVB/ eGFtcGx1IEFURUNDN' AwEHA0IABNV2Er7PN/ oh71rI71QICbaHmeI DgYOVR0PAQH/BAQDA BgNVHSMEGDAWgBQt4( AiB1+t1IGDNJdfts/ HtXAE+B6inLWIE54ZI DD CERTIFIC/	ICATE gIRRLaEewj2iY835Z0gzDi4TwEwCgYIKoZIzj0EAwIwQzEd XBsZS8Pcmdhbm16YXRpb24xIjAgBgNVBAMMGUV4YUIwbGUg 30gQ0EwIBCNMTYWNzESMjAwMDAwMhgPOTKSOTEyHZEyMzUS AOMFEV4YWIwbGUgT3JnYVSpemf0aW9UMSYWJAYDVQQDDBJF TA4QSBTaWduZXIgQKFGMz8ZMBMGByqGSM49AgEGCCqGSM49 d420mJicxJZGN6x3pAv0+4+sQYep0oHctecyX03umqCyu7X M55kNcoNJr1XQajZjBkMBIGA1UdEwEB/wQIMAYBAFCAQAw gKEMB0GA1UdDgQWBBQt4Q3KA3xzmBjP76cQDKQIoGv0gDAf Q3KA3xzmBjP76cQDKQIoGv0gDAKBggqhkjOPQQDAgNHADBE Az5x2syLeXk/pKD5eI+mATM6K6xzgIgG8rd71E3CsRcRGsB DRVnIF9rNs=	<u>insight.js:1001</u>
signerSend:	3, 4, 5	insight.js:1065
3EE3EB1061EA74A07	Lugcuzusucgig/19950004/00/0522500000599220/PEDEE0/488_DBF512BECF55DE50D28262/3 72D79CC97D37BA6A82CAEED7A21EE5AC8EF540909B687FA610CE7990D728589AE55D06)	1739100E010E307FD
The array is 201 b	bytes. Writing 64 bytes at a time	insight.js:1077
signerState: Init	6 Hash to be signed of the Production Signer Cert	insight.js:1113
signerReply: 00 44	41A2C929F499C12038DA357DBE486A78FDF218526027518FC23FAE4C72D3E61	<u>insight.js:1114</u>
rootSend: aws:si(	03,441A2C929F499C12038DA357DBE4B6A78FDF218526027518FC23FAE4C72D3E61)	insight.js:905
The array is 76 by	ytes. Writing 64 bytes at a time	insight.js:917
rootState: SignSig	gner 7 Production Signer Certificate Signature	insight.js:953
rootReply: 00(524B7FDD3BA8370 AD9A67D20890975380	BDFBASCD38130EC55EEC6B9F39DD93657747B09BD6DFEE8B9944798543FF912EB12A72F915CD230 B2)	insight.js:954 DF973EC7D5A71076C
signerSend: aws:ss(01,524B7FDD A71076CAD9A67D2089	D38A8378DF8A5CD38130EC55EEC689F39DD936577478098D6DFEE889944798543FF912E812A72F 909753882)	<u>insight.js:1065</u> 915CD23DF973EC7D5

Figure 7-14. Registering the Production Signer and Verification Code Certificate into AWS IoT part 1

۹	1	Elements	Network	Sources	Timeline	Profiles	Resources	Audits	Console		>_	♣ 🖳	×
$\otimes$	Y	<top frame=""></top>	▼ 🔲 P	reserve log									
signerState:         GetSignerCert         insight.js:1113           signerReply:         insight.js:1114         00(232F42415144416748454D42304741315564446751574242544738465752777566794D36527134694E676144624A48375670556A41660           A42674E5648534D4547444157674251743451334B4133787A6D426A5037366351444B51496F4776306744414B42676771686B6A4F5051514           441674E54841444245044169425353332F644F3667337666756C7A54675444735665374775664F6432545A586448734A7657332B364C6D514           96752486046512F2B524C72457163766B560A7A53506666C7A3748316163516473725A706E306769516C314F4C493D0A2D2D2D2D2D454E442           043455254494649434154452D2D2D2D2D0A)									•				
s	signe	rSend: aws:;	g(02,00)							in	sight.	js:1065	
5	signe	rState: Get	VerifyCer	t						in	sight.	js:1113	
2 2 2 1 8	signe 00(03 34F4C 25842 14D44 30A4D	rReply: 2D2D2D2D2D2D4 4A6F6A425339 735A53425063 684249464E7( 6A4D314F5459	24547494E 92F464841 36D646862 35A32356C 535576A42	204345525 584779774 6D6C36595 636941774 714D52307	4494649434 9774367594 852706232 D4441774D4 747775944	4154452D2 494B6F5A4 34784A6A4 434158445 5651514B)	2D2D2D2D0A4 497A6A30454 416B42674E5 5445324D446 )	D494942 1774977 642414D 3784F54	31444343415875674 527A45640A4D42734 4D485556345957317 49774D4441774D466	<u>in</u> 17749424167495 74131554543677 7624755670A515 F59447A6B354F5	<u>sight</u> 255582 755529 652469 468784	<u>.js:1114</u> 2F64677 5868686 51304D3 4D6A4D7	
5	signe	rSend: aws:	g(02,01)							in	sight.	js:1065	
5	signe	rState: Get\	/erifyCer	t						in	sight.	js:1113	
5 0 1 1 0	signe 00(13 54756 54D7A 14249 0486C	rReply: 444252466543 6D4F574E6A5/ 6B794D6D4D33 4F360A57646/ 595A355A2F5/	746746347 A44426A5A 35A575A69 A49705A5A A63762F32	786C49453 444A6B4D3 5A5759774 6475704E6 610A6E676	9795A3246 2526A4F44 E7A52685A F35447439 672753559	756158700 45774E7A4 54425A4D4 417757560 5C4C4C72)	5864476C766 45354F544D3 424D4742797 54396A704A7( )	26A464A 44D4751 147534D 04C354D	4D456347413155450 304E7A42694E324D7 34394167454743437 4A486372453746383	<u>in</u> 0A417778414E6A5 7A4D6A49314D474 7147534D3439417 2353877746B514	<u>sight</u> A6D590 A690A4 745484 D70450	<u>is:1114</u> 5A526B4 4D47493 4130494 57314E7	
5	signe	rSend: aws:	g(02,02)							in	sight.	js:1065	
5	signe	rState: Get\	/erifyCer	t						in	sight.	js:1113	
9 6 7 7	signe 00(23 07457 85359 73238	rReply: 2B535874706/ 6C534D416F4 632B7369367 4757412B6730	A73756A49 743437147 20A644934 D3D0A2D2D	7A41684D4 534D34394 6D4D47506 2D2D2D454	238474131 2414D4341 64169412F E44204345	556449779 3063414D4 524266649 525449464	51594D42614 45514349434 54794B72736 494341544521	1464D62 1525834 A755943 D2D2D2D	77565A4843352F497 4948336D6F6963305 51486944622F68566 2D0A)	<u>in</u> 7A7047726949324 316A312B7656324 38703230494C772	<u>sight.</u> 26F4E7 A4B355 F74410	<u>.js:1114</u> 736B660 5774434 53326B4	
S N N C N N E E E E Z Z Z Z	signerCert:       insight.js:1495         MIIB8DCCA2egAwIBAgIRUX/dgxoLlojB59/FHAXGywEwCgYIKoZIzj0EAwIwQzEd       MBSGAUECgwURXhhbXBsZSBPcmdhbmI6YXRpb24xIjAgBgNVBAMMGUV4YWIwbGUg         QVRFQ0MIMDhBIFJob30gq0EwIBcNMTYwWzE5MjAwMDAwWhgPOTK50TEyMzEyMzU5       MTIAMEcxHTAbBgNVBAOMFEV4YWIwbGUgT3JnYW5pemF0aW9uMSYwJAYDVQQDDBIF         eGFtcGx1IEFURUNDNTA4QSBTahduZXIgMDAwMDBZMBMGByqGSM49AgEGCCqGSM49       AwEHA0IABI06WdjIpZZdupNo5Dt9AwWVd9jpJpL5MJHcrE7F8258wtkQMpEgINPH         VZ52/Zcv/2angfru5Y1LLr+SXtpjsuj2jBKMBIGA1UdEwEB/wQIMAYBAf8CAQAw       Signer and Verification         DgYDVR0PAQH/BAQDAgKEMB0GA1UdDgQWBBTG8FWRwufyM6Rq4iNgaDbJH7VpUjAf       Schwidydegut403KA3xzmBjP76cQDKQIo6v0gDAKBggahkj0PQQDAgNHADBE         ABSS3/d06g3vfulzTgTDSVe7Gut0d2TZXdHsJvW3+6LmQIgRHmFQ/+RLrEqcvkV       25Pf127H1acQdsr2pn0giQ10LI=												
Verifycerc:       Insight.js:1496        BEGIN CERTIFICATE       MIIBIDCCAXugAwIBAgIRUX/dgxOLJojBS9/FHAXGywIwCgYIKoZIzj0EAwIwRzEd         MBsGA1UECgwURXhhbXBsZSBPcmdhbml6YXRpb24xJjAkBgNVBAMMHUV4YW1wbGUg       QVRFQ0MIMbIbIFNp225LciAwMDAwMCAXDTE2MDcxOTIwhDAwMFoYDzk5OTKxMjMx         MjMIOTU5WjBgMR0wGW7DVQQKDBRFe6FtcGxIIE9yZFuaXphdG1vbj5JMEcGA1UE       AwxANjZmvjRkCGVmONNjZDBjZDJxMZRj0DEwNzESOTM4MGQ0DzBiNZMzhjG1INGJ1         MGISMzkyMmM3ZWZiZVWWLRhZTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABIO6       WdjIpZZdupNo5D59AwWVd9jjDJL5MJHcrE7F8258wtkQWpEg1NpH1Y25Z/Zcv/2a         ngru5Y1LLr+SXtpjsujIzAhMB8GAIUdIwQYMBaAFMbwVZHC5/IzpGriI2BoNskf       tuNSMA0GCcqGSM49AMCQCCCRX4HBamcieQj1+vV2JKSWtCKSYc+si6r         dI4mMGPfAiA/RBfdTyKrsjuYCQHiDb/hVhp20ILw/tAc2kG28GWA+g==      END CERTIFICATE													
-	igne	rSend: aws:	n(03)		_		aller of			in	sight.	js:1065	
2	Signi	ng Module Re	egistrati	on Succes	sful!	rodu	iction S	Ignei		in	sight.	js:1517	
5	signe	rState: Publ	(ey			Regist	ration	Succ	essful	in	sight.	js:1113	
5 6 6	signerReply: <u>insight.js:1114</u> 00(83BA59D8C8A5965DBA9368E43B7D03059577D8E92692F93091DCAC4EC5F36E7CC2D910329120D4DA4795867967F65CBFFD9A9E07EBBB9 6252CBAFE497B698ECB)												

Figure 7-15. Registering the Production Signer and Verification Code Certificate into AWS IoT part 2

During the registration process with AWS IoT the GUI will also enable auto registration (JITR), functionality for our production signer CA certificate as shown below in the Figure 7-15.

![](_page_33_Figure_1.jpeg)

Figure 7-16. Activating the Production Signer CA certificate and enabling auto-registration for JITR on the GUI

After we have successfully registered our Production Signer Certificate into AWS IoT we can go and access our account and verified the registration.

![](_page_33_Figure_4.jpeg)

Figure 7-17. Production Signer Registration in AWS IoT

![](_page_33_Picture_6.jpeg)

#### 7.3 Just-In-Time Registration of the AWS IoT device

In the previous section we showed how use-your-own-certificate (**BYOC**), of AWS IoT will allow to use device certificates signed by a production signer, to connect and authenticate with AWS IoT.

In this section we will now use the Just-In-Time Registration support from AWS IoT. With JITR we will eliminate the need to manually registered any device that was signed by the Production Signer and turn this into an automated process.

To enable an AWS IoT device for JITR, the following steps need to take place

- Create, register and activate a CA certificate that will be used to sign your device certificate (like we did in the previous section).
- Enable auto-registration of certificates (like we did in the previous section).
- Create device certificates signed by your CA and install them on your device
- Create and attach a rule with an AWS Lambda action that activates the certificate and then creates ad attaches policies to the certificate.
- Connect to AWS IoT using the device certificate

#### 7.3.1 AWS IOT TLS and JITR Hands- On

The objective of this hands on will be to demonstrate the process that involves the JITR functionality of AWS IoT.

![](_page_34_Picture_11.jpeg)

#### For this portion you must have had completed the previous section successfully

We will be using the following setup:

- AWS IoT device compose of:
  - o ATSAMG55
  - Winc1500 (updated to FW 19.4.4)
  - CryptoAuthXplained Pro
- Secure Insight of Things GUI
- Root Module
- Signer Module

![](_page_35_Figure_0.jpeg)

Figure 7-18. ATSAMG55 AWS IoT device

The **AWS Zero Touch Secure Development Kit** comes with a micro USB cable. Connect one end of the USB cable to the **USB Target** connector in ATSAMG55 setup, as shown in Figure 7-19.

![](_page_35_Picture_3.jpeg)

Figure 7-19. Connecting the USB cable to your ATSAMG55 setup.

Below is the sequence diagram that we will follow for AWS IoT JITR. Here we assume that the process for registering your Production Signer CA certificate and activating the JITR functionality has been completed.

![](_page_36_Figure_1.jpeg)

Figure 7-20. AWS IoT device JITR process

When a device attempts to connect with it's a certificate that it is not known to AWS IoT but was signed by a Signer CA certificate that was registered with AWS IoT, the device certificate will be auto-registered by AWS IoT in a new PENDING-ACTIVATION state. This means that the device is certificate was auto-registered but is not active. This is only controlled by AWS IoT.

The change of status of from PENDING-ACTIVATION to ACTIVE can be done by means of an AWS Lambda action on the registration topic that will activate the certificate, create and attach a policy to it.

For more information about creating a Lambda function and creating and attaching a policy, visit the AWS IoT site bellow.

https://aws.amazon.com/blogs/iot/just-in-time-registration-of-device-certificates-on-aws-iot/

For this hands-on below you can find the Lambda function we are using along with the policy that we are creating and device activation in the Appendix section of this document.

![](_page_37_Picture_0.jpeg)

#### Registering the AWS IoT Device Certificate Using the GUI

Connect your Root Module, Signer Module and AWS IoT device to your desktop, then open the Secure Insight on Things GUI.

Before we proceed we need to make sure that all of our devices (ATSAMG55, Root Module, Signer Module), are being detected. If they are, you should see them enumerated with in the GUI as shown below in Figure 7-21.

Secure Insight on Things
Microchip Insight on Things View Thing Help
Secure Insight on Things Setup
REGISTER SIGNER
PREPARE AWS THING
RE-SCAN CONNECTIONS
AWS Root AT88CKECCROOT
AWS Signer AT88CKECCSIGNER
AWS Thing ATSAMG55 (AWS v1.0.6)
Copyright © Microchip Technology, Inc. Microchip Secu Insight Application v1.0.0

![](_page_37_Figure_5.jpeg)

![](_page_37_Picture_6.jpeg)

If you have all of the devices connected to your desktop and still do not see all of them showing up, go to **View** menu and select **Reload**.

In order to see the log of the Production Signer registration into AWS IoT, we will use as aid the console view with in our GUI. To enable this view, follow the next steps:

- Open the View menu.
- Select Toggle DevTools from the drop down menu.

💿 Secure Insight on Things						-	×
Microchip Insight on Things	ïew Thi	ng Help	p				
Secure Insight (	Thing	Shadow					
Occure maight (	Reload						
	Toggle	DevTool	s				
REGISTER SIGNE	R						
PREPARE AWS THI	NG						
RE-SCAN CONNECT	IONS						
AWS Root							
AT88CKECCROOT							
AWS Signer							
AT88CKECCSIGNER							
AWS Thing							
ATSAMG55 (AWS v1	.0.6)						

Figure 7-22. Enabling the Console view of the Secure Insight on Things GUI

Press the "**Prepare AWS Thing**" button. This will initiate the Device Registration which we previously discussed in the following order.

- 1. Secure Insight GUI will request the IoT Device to build its certificate.
- 2. Secure Insight GUI will request the Signer Module to sign the TBS Hash.
- 3. **IoT Device** will save the Device Signature and Device Certificate.
- 4. Secure Insight GUI will send the access point SSID and Password to IoT Device
- 5. **IoT Device** will save the access point SSID and Password in ECC508.
- IoT Device will stablish a TLS connection with AWS IoT using ECC508 to harden TLS session
- 7. AWS IoT will publish a registration message and set IoT Device in pending activation
- 8. AWS IoT will disconnect Client (IoT Device)

9. **IoT Device** will stablish a TLS connection with **AWS IoT** using ECC508 to harden TLS session

10. JITR has been successful and IoT Device is connected to AWS IoT

; View Thing Help	
🔍 🕅 Elements Network Sources Timeline Profiles Resources Audits Console	>_   🏶 🖳   ×
🛇 🗑 <top frame=""> 🔻 🔲 Preserve log</top>	
Found Hid Devices [object Object],[object Object],[object Object],[object Object]	insight.js:525
Microchip Kit Detected: \\?\hid#vid_04d8&pid_0f31#7&7448882&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}	insight.js:535
Microchip Kit Detected: \\?\hid#vid_04d8&pid_0f30#7&309403d4&0&0000#{4d1e55b2-f16f-11cf- 88cb-001111000030}	<u>insight.js:535</u>
signerSend: aws:p(03)	insight.js:1065
comName: COM6	insight.js:608
pnpId: USB\VID_03EB&PID_2404\6&32A5DFEC&0&1	insight.js:609
manufacturer: ATMEL, Inc.	insight.js:610
signerState: PubKey	insight.js:1113
signerReply: 00(30CF488CC751C413A61A43B4AD2B8AA744C7A394CBD55E7BD53C17B02C2F843300E91433B77C8E4E0E94F3A776 4CA97E6A0829F5E3331FB27A)	<u>insight.js:1114</u> 009132F44AA59C4
signerSend: aws:g(01,00)	insight.js:1065
signerState: GetSignerCert	insight.js:1113
Signer Reply: 00(032020202020424547494E20434552544946494341544520202020200A4D49494238544343415A65674177494 72595060384A593944536F6F544F3756676541457743675949486F5A497A6A304541774977517A45640A4D4273474 552586868625842735A534250636D6468626D6C365958527062323478496A416742674E5642414D4D475556345957 5156524651304D314D44684249464A7662335167513045774942634E4D5459774E7A45354D6A41774D44417757686 445794D7A45794D7A55350A4E546C614D4563784854416242674E5642416F4D46455634)	1151900-15-1114 241674952574868 113155454367775 23177624755670A 37504F546B354F5
signerSend: aws:g(01,01)	insight.js:1065
signerState: GetSignerCert	insight.js:1113
signerReply: 00(13595731776247556754334A6E59573570656D4630615739754D5359774A41594456515144444231460A654746 465552554E444E54413451534254615764755A5849674D4441774D44425A4D424D4742797147534D3439416745474 90A41774548413049414244445053497A485563515470687044744830726971644578364F55793956656539553846 414F68554D3764386A68344F0A6C504F6E646768544C3053715763524D7158357143436E31347A4D66736E716A5A6 1315564457745422F7751494D41594241663843415141770A4467594456523050415148)	<u>insight.is:1114</u> 5746347786C4945 13437147534D343 53741734C34517A 5A426B4D4249474
signerSend: aws:g(01,02)	insight.js:1065
signerState: GetSignerCert	insight.js:1113
<pre>signerReply: 00(232F4241514441674B454D423047413155644467515742425434636C566F34627673474D384F6E4A546534394E 41660A42674E5648534D45474441576742514F724E6E6452706673387544514C415A3058614235385A49377A44414 A4F5051514441674E49414442460A4169417979443456514F5151764F6A6D7933366C654B3772356B396F617A6466 584548706D4C774968414B32624C64503944562F33457361500A337A6A636B6F6D372F6157736F785371676168514 90A2D2D2D2D2D454E442043455254494649434154452D2D2D2D2D2D2DA)</pre>	<u>insight.is:1114</u> 71363964286A6A 84267677168686 576495342753835 52433073375A714
thingSend: aws:i(30CF488CC751C413A61A43B4AD2B8AA744C7A394CBD55E7BD53C17B02C2F843300E91433B77C8E4E0E94F3A 9C44CA97E6A0829F5E3331FB27A)	<u>insight.js:738</u> A77609132F44AA5
thingState: Init <b>2</b>	insight.js:777
thingReply: 00(BE375253DA226F4666D64AB7DCB54D9ACADA3DFA4CD013B279042F5D9D48E1E1)	insight.js:778
signerSend: aws:si(03,BE375253DA226F4666D64AB7DCB54D9ACADA3DFA4CD013B279042F5D9D48E1E1)	insight.js:1065
The array is 76 bytes. Writing 64 bytes at a time	insight.js:1077
signerState: SignThing	insight.js:1113

![](_page_39_Figure_3.jpeg)

![](_page_39_Picture_5.jpeg)

Q	9		Element	s I	Network	Sources	Timeline	Profiles	Resources	Audits	Console	>_ 券 旦,	×
0	T	<	top fram	e>	▼ □ P	reserve log							
signerReply: 00(0DD7EBA62903827441A029DBCA1C03BAAFB33082685D76CE82F9ED0D332FEC789FE9C3AA7F7F14735C2CCD8776806BFC3EE2DCCF E400B79DEC0D5472398D47C1)								*					
	thin aws: EE2D	ngSe ss( )CCF	nd: 03,0DD7 E400B79	EBA6 DEC	52903827 3D547239	441A029DE 8D47C1)	CA1C03BAA	FB3308268	35D76CE82F9	ED0D3321	FEC789FE90	<u>insight.is:738</u> 3AA7F7F14735C2CCD8776806BFC3	
	thin	ngSt	ate: Sa	veS	ig							insight.js:777	
	thin	ngRe	ply: 00	()								insight.js:778	
	thin aws: 4)	sh(	nd: 6133616	4610	5b686933	696379763	92e696f74	2e75732d7	76573742d3	22e616d0	617a6f6e61	<u>insight.js:738</u> 77732e636f6d,4f73636172496f5	
	thin	ngSt	ate: Sa	veHo	ostInfo							insight.js:777	
	thin	ngRe	ply: 00	()								insight.js:778	
	thin	ngSe	nd: aws	:g(0	03,00)							insight.js:738	_
	thin	ngSt	ate: Ge	tThi	ingCert							insight.js:777	
	00(2 506A 2586 5652 8354 7561 2615 4F2B 4523 6A50 2426 3267 D2D2	2D2D 535 868 465 465 746 746 405 405 440 6745 407 200 200 477	2D2D2D4 44B6659 6258427 1304D31 6B784D6 0686447 6C4D466 6304E43 2F4C6A7 4707955 6E62796 6385075 )	2454 6F63 35A9 4D44 A4D7 6C76 8774 3833 9567 3875 3379 8774 4C63	47494E20 34F43486 53425063 46842494 780A4D6A 5626A456 45775948 349736F4 79617548 55054617 44757128 37A28514	434552544 D30484A4E 606468626 64E705A32 44D314F545 84D423847 486F5A497 A6F6F5735 636948334 57658666F 77A0A4D494 174353373	946494341 674977436 056265958 356663694 413155456 A6A304341 68414C693 F4F694461 347743675 A72585865 445652794	54452D2D2 75949486f 527062322 1774D4441 434D52307 A4177755 515949486 667464157 4D6A4D434 949486F54 4F6776667 F59314804	2D2D2D0A4D4 5A497A6A30 44784A6A416 1774D434158 35258686822 355A497A6A3 76D6165624D 15774877594 4497A6A3045 7444544D763 377513D3D0A	9494272 45417749 842674E 4454453 651514B 5842735/ 0440A41 33584A28 45652300 4177494 7486743 20202020	5443434156 977527A456 5642414040 24D4463784 4442524665 4534242564 5163445167 8315772454 5442426777 4534141775 4951436636 020454E442	4F6741774942416749526248494D 440A4D42734741315545436777555 44855563459573177624755670A51 4F5449774D4441774D466F59447A6 54746746347786C494539795A3246 455644517A55774F4545675247563 7414530345868494568337A4E3064 4A656666680A43636F45503949696 7466F415528484A56614F47373742 525149674466667270696844676E5 5634F7166333855633177737A5964 2043455254494649434154452D2D2	
	MIIB MBsG QVRF MjM1 AwwY AQcD CcoE DpyU MIJr wQ==	-BE BrTC GA1U QOM OTU (RXh QgA P9I J3uP XXb -EN	GIN CER CAVOgAw ECgwURX 1MDhBIF SWjBCMR hbXBsZS E04XhIE idR4J/L TauvXfo OgvntDT D CERTI	TIF: IBAg hhb) NpZ: 0wGv BBVG k3z! jyVy 4wCg Mv7! FIC/	ICATE gIRbHIMP KBsZSBPc 251ciAwM vYDVQQKD EVDQZUWO WØdO+MVØ v@dC+MVØ vauHciH3 gYIKoZIz HgCIQCf6	 mdhbml6YX DAwMCAXDT BRFeGFtcG EEgRGV2ak NC83IsoJo 00iDaMjMC -j0EAwIDSA cOqf38Uc1	CHm0HJKgJ Rpb24xJj/ E2MDcxOTJ x1IE9yZ2F N1MFkwEwy oW5hALi6g EwHwYDVR0 AwRQIgDdf wszYd2gGv	wCgYIKoZJ kBgNVBAMM wMDAwMFo` uaXphdG1 HKoZIzj0( FAWmaebM jBBgwFoAL rpikDgnRE 8PuLcz+QA	Izj0EAwIwRz HUUV4YW1wbG DzkSOTkxMj /bjEhMB8GA1 AQYIKoZIzj 3XJ+1WrEJef J+HJVaOG77B 30ChbyhwDuq At53sDVRyOY	Ed Jg Mx JE ØD nk jP +z 1H		<u>insight.js:844</u>	
	+bin	- EN	ndi awa		(01	-REGIN CE	DTTETCATE			3		incidht ic:738	_
	MIIB MBsG QVRF NTla eGFt AwEH 1POn DgYD BgNV AiAy 3zjc )	Alu AMEc CGX AØI AØI VRØ VRØ VRØ VRØ S CGX AØI AØI AØI AØI AØI AØI AØI AØI AØI AØI	CAZEGAW ECgwURX 1MDhBIF xHTAbBg 1IEFURUI ABDDPSI TLØSQWCI PACH/BA EGDAWgB VQQQVO 7/AWSOX D CERTI	IBA IBA Ibb) Jvb3 NVB/ NVB/ NDN1 ZHU QDA QDA QOrl Jmy3 Sqga FIC/	gIRWHKrY gIRWHkrY (BsZSBPc 3QgQ0EwI AOMFEV4Y FA4QSBTa cQTphpDt (SqCCn14 gKEMB0GA yndRpfs8 361eK7r5 ahQBC0s7 ATE	PR8JY9DSc mdhbml6Y3 BcNMTYwNz W1wbGUgT3 WduZX1gMD KØriqdEx6 zmfsnq12j 1UdDgQWBE wDQLAZØXa k9oazdfv1 ZqI	IGHT IGHT Rpb24x1j4 E5MjAwMD4 JnYW5pemF AwMDBZMB iOUy9Vee9U BKMBIGA11 T4c1V04bv B58ZI7zD4 SBu85XEHp	wCgYIKoZI gBgNVBAM wWhgPOTk5 0aW9uMSYy IGByqGSM49 8F7AsL4Q2 sGM8OnJTe KBggqhkjC mLwIhAK20	IZJØEAWIWQZ NGUV4YW1WbG SOTEyMZEYMZ NJAYDVQQDDB OAgEGCCqGSM 2AOKUM7d8jk MAYBAF8CAQ 249Nq69d+jj DPQQDAgNIAD 5LdP9DV/3Es	Ed Jg J5 IF 49 40 Aw Af BF BF		<u>INSIGUE, 15:730</u>	•

Figure 7-24. JITR process using GUI part 2

Once the device has been registered in to AWS IoT, we can go into our AWS IoT account and verify it, like the Figure below.

AWS IOT	Resource	Resources   MQTT Client   Tutorial   Settings   0 notifications					
(All) 0/0 things 0/0 thing types 1/1 rules 7/7 CAs	Select all Actions -	Learn more <u>Detail</u> X					
4/4 certificates 1/1 policies	First Previous 1 Next Last	Certificate ARN arr:aws:iot:us-west-2:29918333782 6:cert/4bda302fcaf032bf10/97/5437 cd0684d2f8901d89a1d79ebca5e425 343037					
on 884f17adf204e	b4e2257edbc1 aca93d0b34	Status ACTIVE					
ENABLED ACTIVE	ACTIVE	Issuer CN=Example ATECC508A Signer 0000,O=Example Organization					
	\$ D	Subject CN=Example ATECC508A Device,O=Example Organization					
		CA ac540fd71366a87c25c94063303 d2c1c71da5c9598d3d757f69c23 bedec0890f					
29bd683d615f 28158d5bca25 191a15a35011 a9f3cd062275 bff64ccee24a7 942312dbb96c	8c4c478c9c3b da336a5ffe4ba	Created date Sep 20, 2016 10:06:42 AM -0600					
c155c2ba28 7f06bff3032 bacf29a7da7	d2f4e42878f	Effective date Jul 19, 2016 2:00:00 PM -0600					
ACTIVE ACTIVE ACTIVE	PENDING_ACT IVATION	Expiration date Dec 31, 9999 4:59:59 PM -0700					
	\$ D	Select all Detach 🛓					
8eee04e957a8         f8669e2c8d39         ae540/d71366           edab35f2a43d         ddd7232x8049         a87e25e94063           b1/d602734         7565d6660         303d2e1c71	Device Certificate 4bda302/caf03 2bt10/97/fa43 7cd0684d2f JITR Successfuly	thingPolicy					
PENDING_ACT ACTIVE ACTIVE	ACTIVE completed						
	S .	through AWS Lambda action					

Figure 7-25. JITR successfully completed

![](_page_41_Picture_2.jpeg)

### 8 Appendix

#### 8.1 Lambda Function, Policy Attachment and Device Activation

```
/**
This node is Lambda function code creates and attaches an IoT policy to the certificate
registered. It also activates the certificate. The Lambda function is attached as the rules engine action to
the registration topic aws/events/certificates/registered/<caCertificateID>
event JSON Structure:
{
 "certificateId": "<certificateID>",
 "caCertificateId": "<caCertificateId>",
 "timestamp": <timestamp>,
 "certificateStatus": "PENDING_ACTIVATION",
 "awsAccountId": "<awsAccountId>",
 "certificateRegistrationTimestamp": "<certificateRegistrationTimestamp>"
}
**/
var AWS = require('aws-sdk');
var region;
var iot;
var accountId;
var certificateARN:
var certificateId;
var thingPolicy;
const awsPolicyName = 'thingPolicy';
// Delay time tracking
var eventTime;
exports.handler = function(event, context, callback)
{
// Step 1: Create the policy.
// Step 2: Attach the policy to the certificate
// Step 3: Activate the certificate.
//Optionally, you can have your custom Certificate Revocation List (CRL) check logic here and
//ACTIVATE the certificate only if it is not in the CRL .Revoke the certificate if it is in the CRL
// Capture the event time & delay for Lambda execution
  var currentTime = (new Date()).getTime();
  eventTime = event.timestamp;
  var eventDelay = currentTime - eventTime;
  console.log("Lambda event delay: " + eventDelay);
```

# **Atmel**

```
// Replace it with the AWS region the lambda will be running in region = "us-west-2";
// Get the AWS account ID
accountId = event.awsAccountId.toString().trim();
// Create the lot object
iot = new AWS.lot({'region': region, apiVersion: '2015-05-28'});
certificateId = event.certificateId.toString().trim();
// Construct the ARN for the Thing certificate
certificateARN = `arn:aws:iot:${region}:${accountId}:cert/${certificateId}`;
// Create and attach the thingPolicy
awsCreateSimplePolicy();
};
function awsCreateSimplePolicy()
{
  // Step 1: Create the policy
  // Policy definition
   var policy = {
          "Version": "2012-10-17",
          "Statement": [{
                    "Effect": "Allow",
                    "Action":["iot:*"],
                    "Resource": ["*"]
          }]
  };
  // Create the policy
  iot.createPolicy(
  {
     policyDocument: JSON.stringify(policy),
     policyName: awsPolicyName
  }, (err, data) =>
  {
     // Log the delay for the createPolicy() callback
     var currentTime = (new Date()).getTime();
     var callbackDelay = currentTime - eventTime;
     console.log("awsCreatePolicy() Delay: " + callbackDelay);
```

![](_page_43_Picture_2.jpeg)

```
// Ignore if the policy already exists
     if (err && (!err.code || err.code !== 'ResourceAlreadyExistsException'))
     {
        console.log(err);
        return;
     }
     // Set the thingPolicy to the return data
     thingPolicy = data;
     // Step 2: Attach the policy to the certificate
     awsAttachPolicy();
  });
}
function awsAttachPolicy()
{
// Step 2: Attach the policy to the certificate
// Attach policy to certificate
iot.attachPrincipalPolicy(
{
  policyName: awsPolicyName,
  principal: certificateARN
}, (err, data) =>
{
  // Log the delay for the attachPrincipalPolicy() callback
   var currentTime = (new Date()).getTime();
   var callbackDelay = currentTime - eventTime;
   console.log("awsAttachPolicy() Delay: " + callbackDelay);
  // Ignore if the policy is already attached
  if (err && (!err.code || err.code !== 'ResourceAlreadyExistsException'))
  {
  console.log("Failed to attach Policy to \"Thing\" certificate\n" + err);
  return;
  }
  // We've attached the policy, now activate the certificate
  awsActivateThing();
  });
}
```

```
function awsActivateThing()
{
  //Step 3: Activate the certificate.
  //
                               Optionally, you can have your custom Certificate Revocation List (CRL) check logic here and
  //
                               ACTIVATE the certificate only if it is not in the CRL .Revoke the certificate if it is in the CRL
  iot.updateCertificate(
  {
     certificateId: certificateId,
     newStatus: 'ACTIVE'
  }, (err, data) =>
  {
     // Log the delay for the updateCertificate() (activate) callback
     var currentTime = (new Date()).getTime();
     var callbackDelay = currentTime - eventTime;
     console.log("awsActivateThing() Delay: " + callbackDelay);
     if (err)
     {
       console.log("Thing activation failed.");
     }
     else
     {
       console.log("Thing activated successfully.");
     }
  });
}
```

![](_page_45_Picture_2.jpeg)

#### 9 License Information

![](_page_46_Picture_1.jpeg)

www.atmel.com

MICROCHIP Atmet Enabling Unlimited Possibilities **f in 8 D W** 

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA T: (+1)(408) 441.0311 F: (+1)(408) 436.4200

© 2014 Atmel Corporation. / Rev.: Error! Reference source not found. - Error! Reference source not found.: Error! Reference source not found.

Atmel<sup>®</sup>, Atmel logo and combinations thereof, Enabling Unlimited Possibilities<sup>®</sup>, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM<sup>®</sup>, ARM Connected<sup>®</sup> logo, and others are the registered trademarks or trademarks of ARM Ltd. Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death ("Safety-Critical Applications") without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.