Application Note

Implementation of Elliptic Curve Cryptography Authentication on TI Battery Fuel Gauges

Michael Smith, Garry Elder

ABSTRACT

Most Texas Instruments battery fuel gauges support some level of authentication functionality for anti-counterfeit protection of battery packs. This application note summarizes the authentication features and details the process of creating and programming an Elliptic Curve Cryptography (ECC) key pair using the BQ41zxx family of products as an example.

Table of Contents

Trademarks

1 Introduction

Elliptic Curve Cryptography (ECC) is an authentication scheme that exploits the mathematical properties of elliptic curves to generate an asymmetrical private and public key pair. ECC algorithms are available in several different versions such as ECDSA (detailed in FIPS 186-5) and EC-KCDSA. The BQ41Zxx family of TI battery fuel gauges uses the Elliptic Curve Korean Certificate-based Digital Signature Algorithm (EC-KCDSA), implemented based on a paper published by the KCDSA Task Force Team.

The implementation used in the BQ41zxx family of devices provides an EC-KCDSA signature, or response from a challenge, based on the B-233 and uses a SHA-256 algorithm for the hash (detailed in FIPS 183-4). The implementation uses the X and Y coordinates of the public key and padded to the correct length.

The authentication functionality of the gauge is accessed through the SMBus interface using the MaufacturerAccess( ) command detailed in the BQ41z50 Technical Reference Manual. The BQ41z50 can be authenticated by a host device, such as a notebook computer, and the gauge can also authenticate the host to allow the gauge to be re-configured or reprogrammed.

Table 1-1. Summary of SMBus Commands Used

Type	ID	Function	Mode	Access
MAC	0x0034	HostPublicKey( ) Allows read and write of the host authentication public key. Note 1 - Once host auth public key is set, legacy 'two-word unseal' method is IMMEDIATELY disabled. Note 2 - This can be written back to all zeros to disable host auth while in full access mode.	Read/Write	R: S/U/F W: F
MAC	0x0036	GaugeAuthPubKey( ) A single "compressed point" public key for authenticating the device Read returns the key status byte and 30 bytes of the public key (compressed with LSB first)	Read	S/U/F
MAC	0x0038	ProdPrivateKey( ) Used to program the gauge authentication private key (private key 30 bytes + public key compressed point 30 bytes)	Write Only	F
MAC	0x003a	ECC_MAC( ) Used to allow host authenticated unseal commands to run	Read/Write	S/U/F
MAC	0x003c	ECC_R A read return the most recent gauge auth result *r* if available. A writes is for the host to write authentication data to the gauge as part of ECC_MAC( ).	Read/Write	S/U/F
MAC	0x003d	ECC_S A read return the most recent gauge auth result *s* if available A writes is for the host to write authentication data to the gauge as part of ECC_MAC( ).	Read/Write	S/U/F
SBS	0x2f	GaugeAuthentication( ) Used to write challenge to gauge and read the 60-bytes of *r* and *s*.	Read/Write	S/U/F

For more information on the standards, see Section 8.

2 Authentication Scheme Comparison

ECC offers several important differences compared to Secure Hash Algorithm (SHA) authentication schemes such a SHA-1. First, ECC uses asymmetric keys so the host and gauge do not share a single key or secret and a key pair (one public and one private) must be used to authenticate the device. Both schemes can have a challenge length of 20-bytes to provide a random challenge.

One major difference between the two authentication protocols pertains to authentication verification. With SHA, the host can start verification in parallel to the gauge because the host already contains the secret key and random challenge once verification is initiated, while the gauge begins once the gauge has received the challenge. ECC requires the host to wait for the response from the gauge to complete the verification process. However, the host can pre-start certain calculations while waiting on this response.

A second major difference between the two authentication protocols is verifying key programming. When using the same key and challenge then SHA produces the same response. Therefore, the challenge-response pair can be used to verify if the key is programmed correctly. With ECC, the same key and same challenge does not produce the same response. A separate verification function must be implemented to verify the key is programmed correctly.

Table 2-1. ECC v. SHA-1 Authentication Algorithms

	SHA-1 HMAC	ECC
TI Product	BQ40z50 and BQ41z50	BQ41z50
Key Type	Symmetric Key (Shared Secret)	Asymmetric Key (Public and Private Key Pair)
Hash Function	160-bit	256-bit (SHA-2)
Key Length	128-bit	233-bit Key
Authentication Response Time	<100ms	<100ms
Challenge Length	20-byte	8 - 19 byte
Response Length	20-byte	60 bytes (or 2 x 30 bytes)
Deterministic response for a given key and challenge	No	Yes
Verify key programmed without using private data	Yes Use known Challenge-response Pair	Yes Verify function using public key

3 ECC Key Programming Within the Pack Manufacturing Flow

The BQ41z50 contains several sets of keys for differing purposes which are noted in the Table 3-1.

Table 3-1. List of Key Pairs and Secrets

Key	Owner	Use
Host command key (public only)	Customer	Used to authenticate host commands
Gauge Authentication Key (private, public)	Customer	Host generated and programed to BQ41z50 devices. Used for battery-host system authentication.

The Gauge and Host production key pairs must go through a specific procedure to be generated and uploaded to the BQ41z50.

Key generation: Random 233 bit value for private key as 'x', generate public key as (x^-1)G, alternatively, generate a standard ECDSA key, and then (ECKCDSA private key) = (ECDSA private key)^-1

Key programming (Full access only): Write private key (30 bytes, LSB first) || public key (30 bytes, LSB first, compressed) to MAC 0x0038

4 Gauge Authentication Flow of the BQ41z50 Product Family

The authentication process can be performed by the pack manufacturer or assembly house to make sure a complete, authenticated battery pack is constructed. The authentication flow is visualized in the Figure 4-1 diagram using the SMBus ManufacturerAccess( ) (MAC) commands to the gauge.

An overview of the flow is:

1. Host Reads ProdKpub from Gauge
2. Gauge Authentication
   - Host send challenge message to gauge
   - Gauge signs the message with ECC-233
   - Gauge provides r/s to host
   - Host verifies the signature

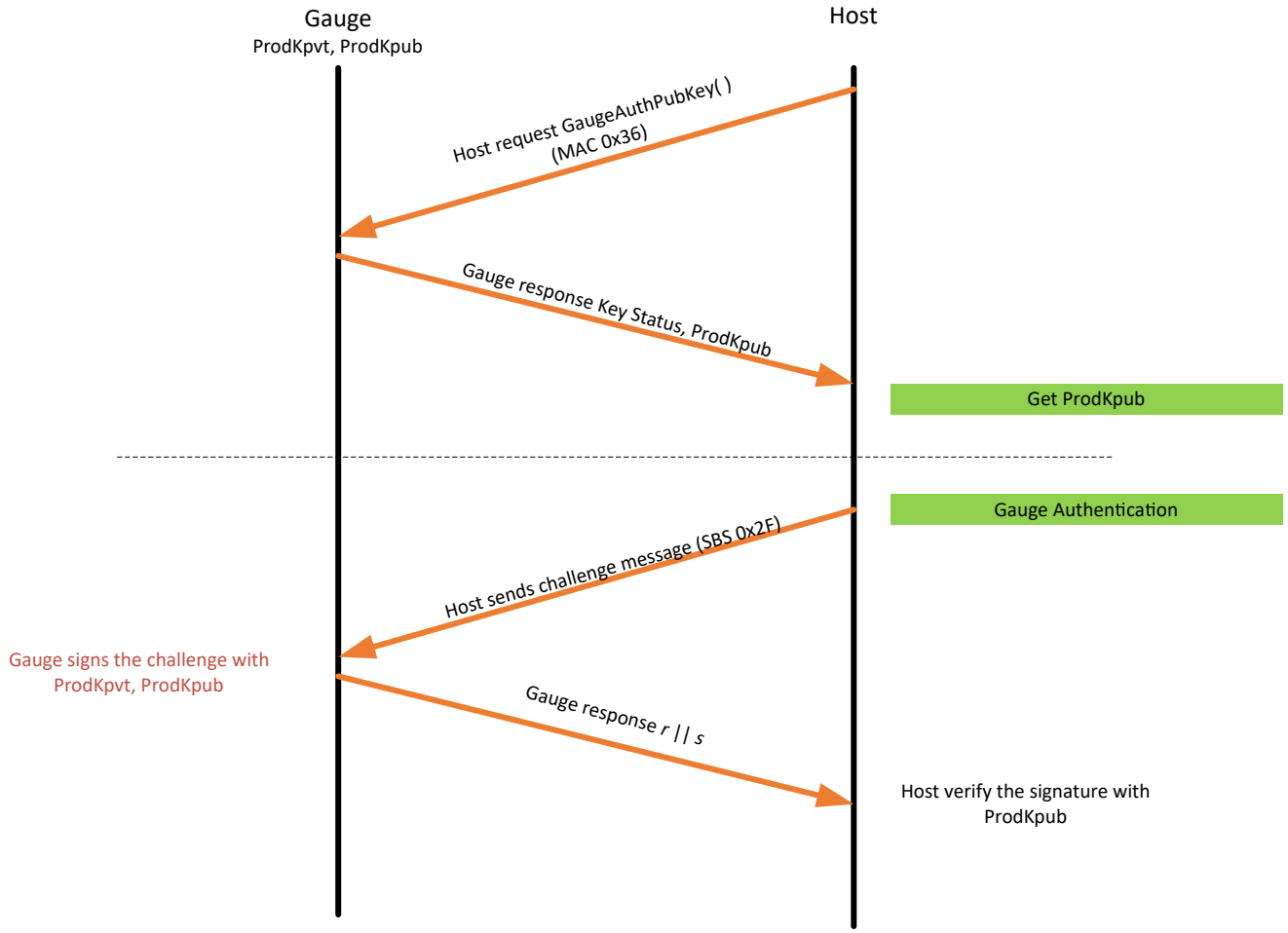


Figure 4-1. Gauge Authentication Overview

A more detailed implementation flow chart is shown in Figure 4-2.

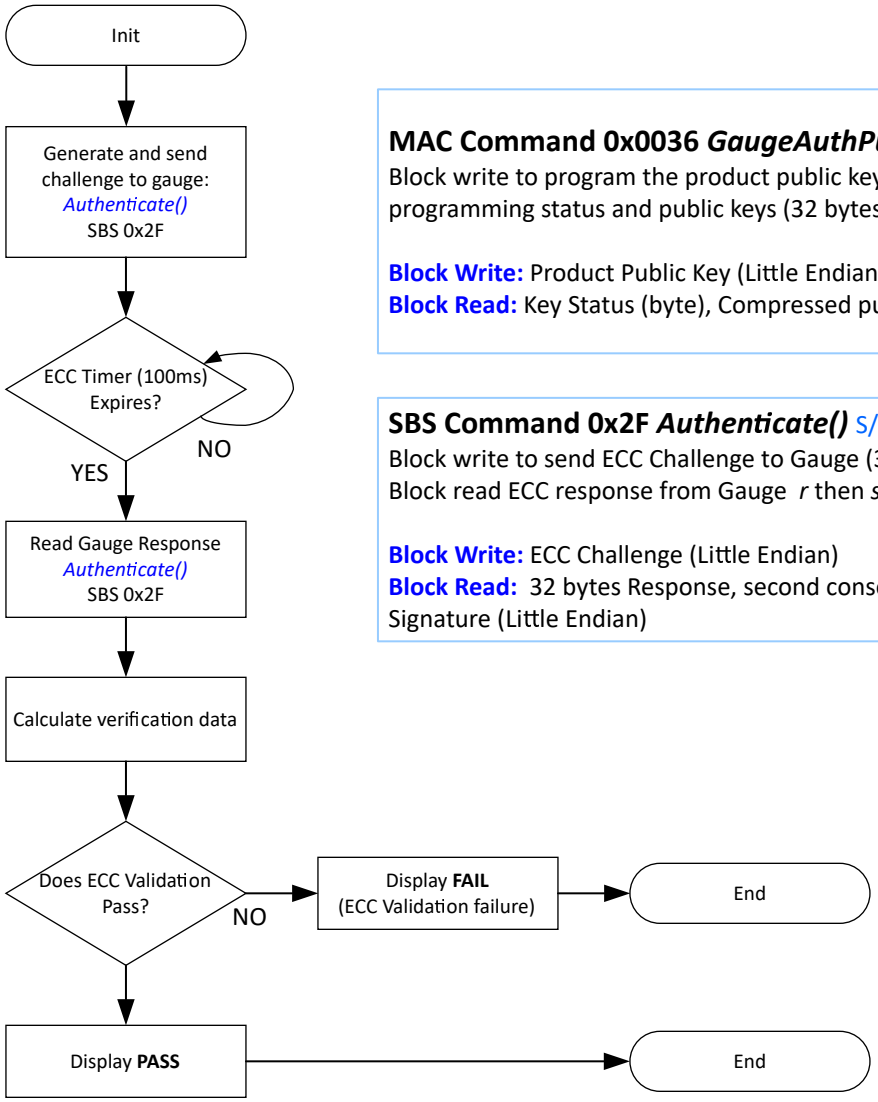


Figure 4-2. Gauge Authentication Flow Chart

5 Host Authentication Flow of the BQ41z50 Product Family

To better secure access to the device the gauge can also be used to authenticate the host allowing the gauge to change security states and unlock the gauge allowing the host to update the gauge. In order to perform this function, the host must first program an authorized public key. The public key must be written when the gauge is in a 'full access' state by sending the 30 byte compressed public key to MAC subcommand 0x0034. This same command must be read to confirm the currently programmed public key for host authentication.

When there is no public key is programmed, the old 'security keys' unseal operation is used. Once a public key is programmed however, the old unseal commands are disabled.

Note

Make sure to read the public key after programming to confirm the value was stored correctly before sending the SEAL command. There is no way to recover a device that in the SEALED state without the corresponding private key.

The procedure must follow the Figure 5-1 diagram using the SMBus ManufacturerAccess( ) (MAC) commands to the gauge.

An overview of the flow is:

1. Host sends MAC subcommand 0x003a with data using one of the following data blocks:
   - To request UNSEAL access action, use data block 0x14, 0x04, 0x72, 0x36
   - To request FULL_ACCESS access action, use data block 0xff, 0xff, 0xff, 0xff
2. Host reads the MAC result using 0x003a to receive the generated 8-byte challenge code.
3. Host combines the 8-byte challenge and the command into a message, and signs the message with the private key generating a 30-byte *r*, and 30-byte*s*.
   - For example, if the gauge generated the challenge 0x12, 0x34, 0x56, 0x78, 0x9a, 0xbc, 0xde, 0xf0, then the full message string to sign for an UNSEAL operation is 0x12, 0x34, 0x56, 0x78, 0x9a, 0xbc, 0xde, 0xf0, 0x14, 0x04, 0x72, 0x36
4. Host writes the resulting *r* and *s* to the gauge.
   - The ECC_R (0x003c) subcommand accepts either the 30-byte *r* value, OR the full 60-byte *r*, *s* value.
   - The ECC_S (0x003d) subcommand accepts the 30-byte *s* value, written after the *r* value has been sent to the gauge.
5. Once the gauge has both *r* and *s*, the gauge validates the signature.
6. If the signature is valid, the action requested by the command is executed.

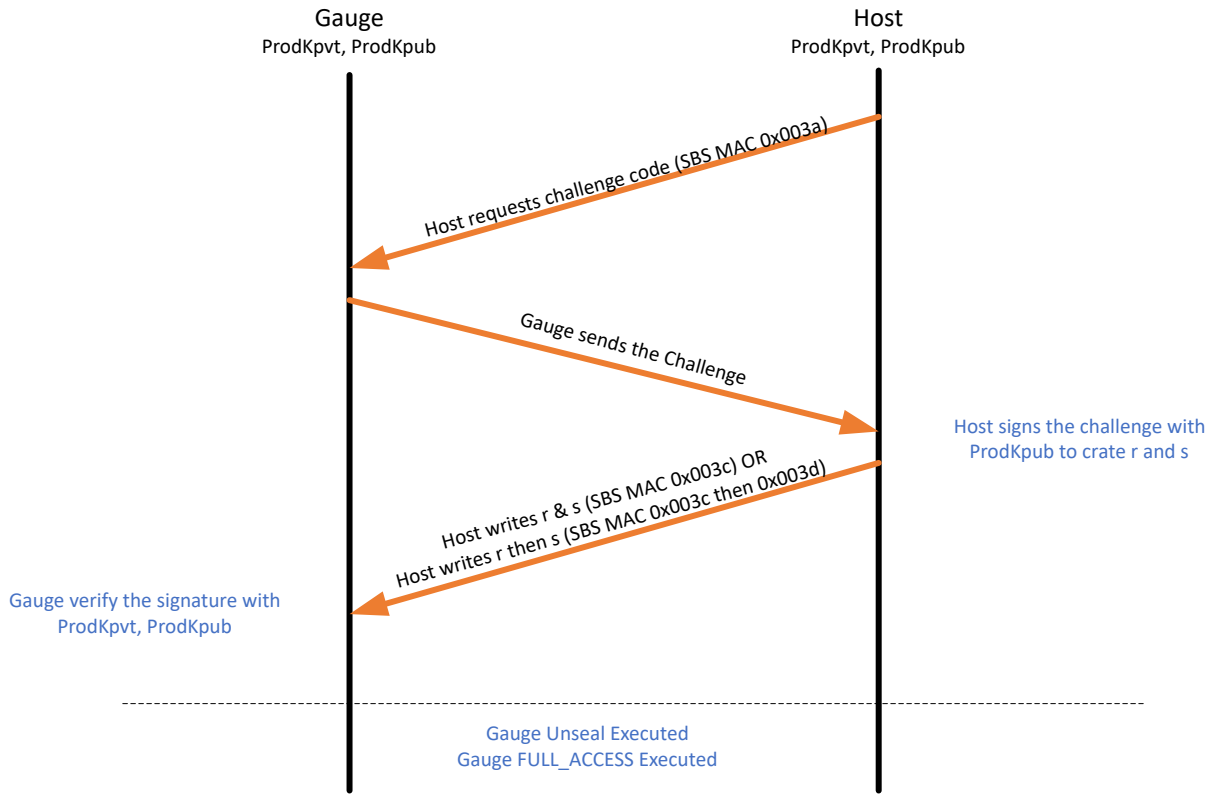
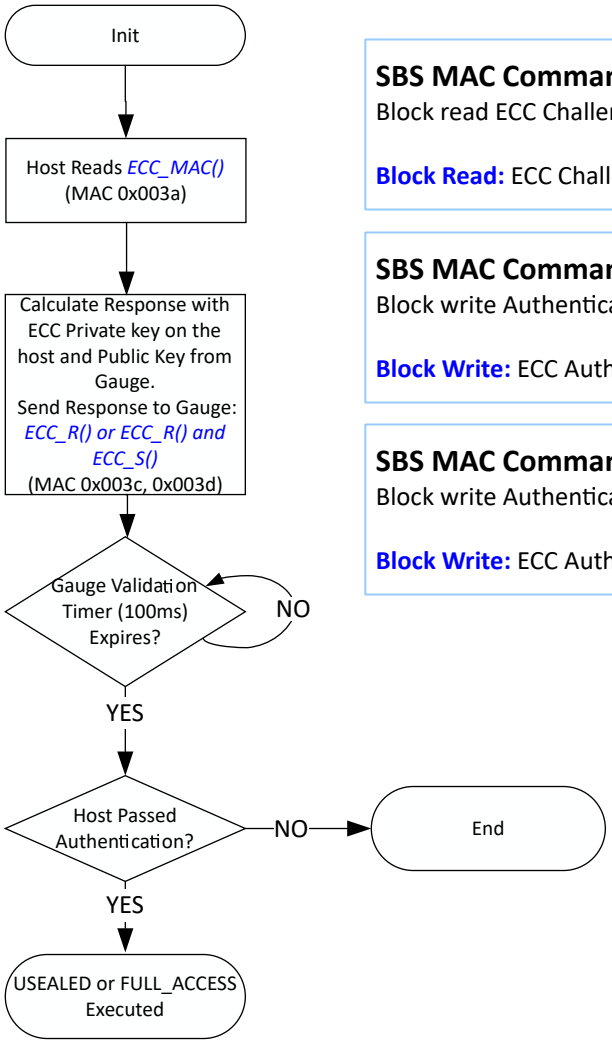


Figure 5-1. Host Authentication Overview

A more detailed implementation flow chart is shown in Figure 5-2.



SBS MAC Command 0x003a ECC_MAC() S/U/F
Block read ECC Challenge from Gauge (8 bytes)

Block Read: ECC Challenge (Little Endian)

SBS MAC Command 0x003c ECC_R() S/U/F
Block write Authentication *r* or *r* & *s* (30 or 60 bytes)

Block Write: ECC Authentication *r* or *r* and *s* data (Little Endian)

SBS MAC Command 0x003d ECC_S() S/U/F
Block write Authentication *s* (30 bytes)

Block Write: ECC Authentication *s* data (Little Endian)

* Command availability under the following Security Modes:
S: Seal Mode
U: Unseal Mode
F: Full Access Mode

Figure 5-2. Host Authentication Flow Chart

6 Authentication Flow in BQSTUDIO

BQSTUDIO is the tool provided by Texas Instruments that is used to communicate with the BQ41z50. For the purpose of the authentication flow covered in this document the *Advanced Comm SMB* tab is the one used. Figure 6-1 notes some of the key user interactions mentioned in the document.

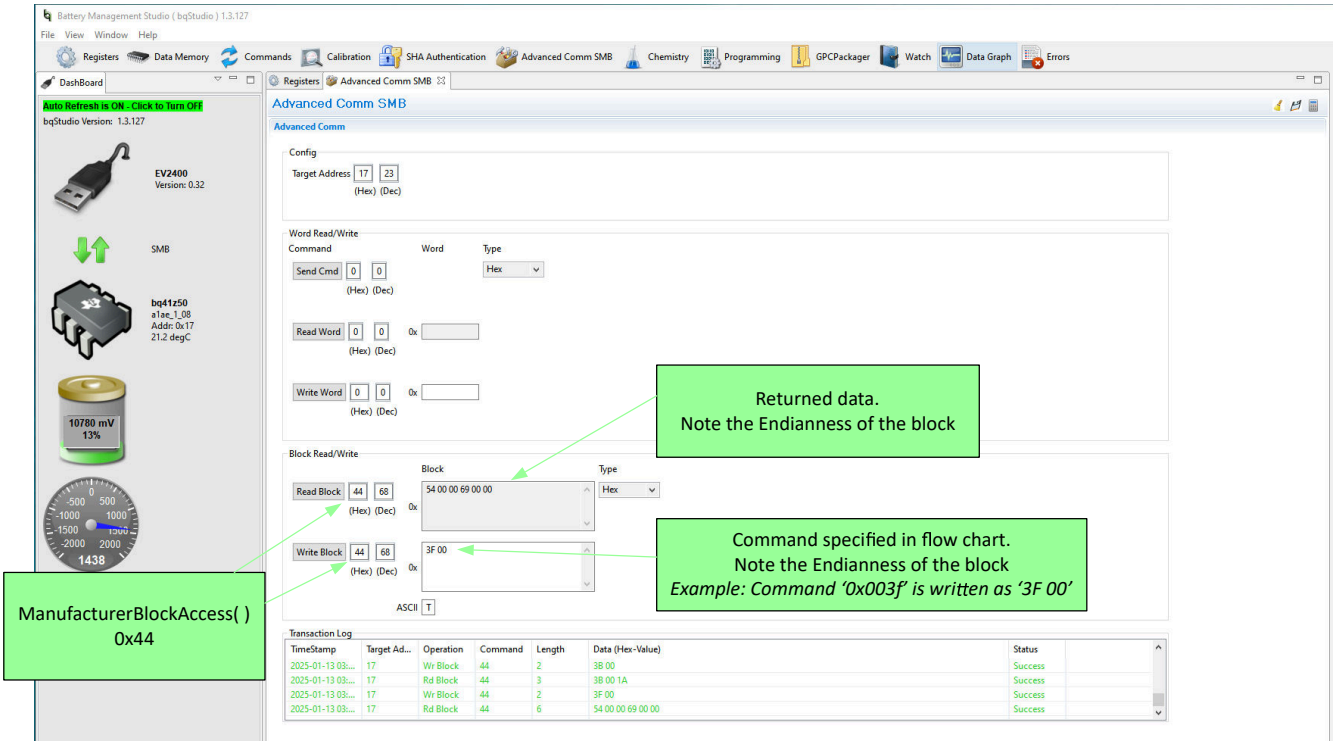


Figure 6-1. BQSTUDIO Advanced Comm SMB Tab

7 Summary

In summary, this application note describes how the authentication keys for the BQ41z50 device are generated, how the key data can be programmed into the device and how a battery enabled with the bqBQ41z50 is used in battery authentication.

8 References

- KCDSA Task Force Team, *The Korean Certificate-based Digital Signature Algorithm*
- National Institute of Standards and Technology, *FIPS 186-5 Digital Signature Standard (DSS)*
- National Institute of Standards and Technology, *FIPS 180-4 Secure Hash Standard (HSS)*
- International Organization for Standardization, (ISO) *ISO/IEC 14888-3:2018 IT Security techniques — Digital Signatures*
- Texas Instruments, *BQSTUDIO Battery Management Studio Software*
- Texas Instruments, *BQ40Z50 1-4 series Li-ion battery pack manager | battery fuel (gas) gauge*
- Texas Instruments, *BQ41Z50 2 to 4 series Li-ion highly integrated battery fuel gauge and protector*

Table 8-1. Algorithm Summary and Notes

Algorithm	Notes
ECDH	#peer_kpub is 32 bytes #self_kpri is 32 bytes #secret is 30 bytes `secret = ecdh(peer_kpub, self_Kpvt)`

PBKDF2HMAC	#Sample python code from openssi
	#Salt is random number that is read from gauge
	#length is 16 bytes for AES-128
	#iterations is 128
	#key is AES-128 key
	
	`from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC`
	`kdf = PBKDF2HMAC(`
	`aglorithm = hashes.SHA256( ),`
	`length = 16,`
	`salt=bytes(salt),`
	`iterations = 128,`
	`backed=backend`
	`)`
	`aeskey = bytearray(kdf.derive(bytes(secret)))`

AES	#iv is read from gauge
	Encrypted ProdKpvt CProdKpve is 32 bytes
	
	`from Crypto.Cipher import AES`
	`cipher = AES.new(aeskey, AES.MODE_CTR, nonce=bytes(iv[0:8]),`
	`initial_value=bytes(iv[8:16]))`
	`CProdKpvt = cipher.encrypt(ProdKpvt)`

ECC	ECC-233

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265