NXP & Security Innovation
Encryption for ARM MCUs

Security Innovation®
THE SOFTWARE SECURITY COMPANY

founded by
PHILIPS

Presenters

- **Gene Carter**- International Product Manager, NXP Semiconductors
  Gene is responsible for marketing of the ARM7 and Cortex-M3 microcontrollers. He has worked in the semiconductors industry for 15 years; 12 of those years with Philips/NXP.

  Gene holds a BSEE from Tufts University and an MBA from the University of Southern California.

- **Peter Jenney** - VP of Product Management, Security Innovation
  Pete leads all product management and strategy for the company's software security products.

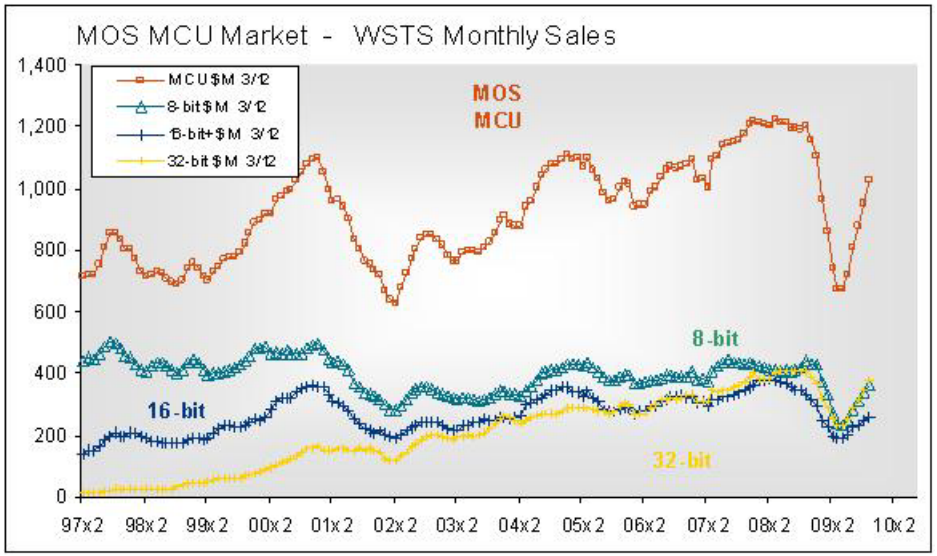
  Pete received his B.A. degree from the University of Massachusetts.

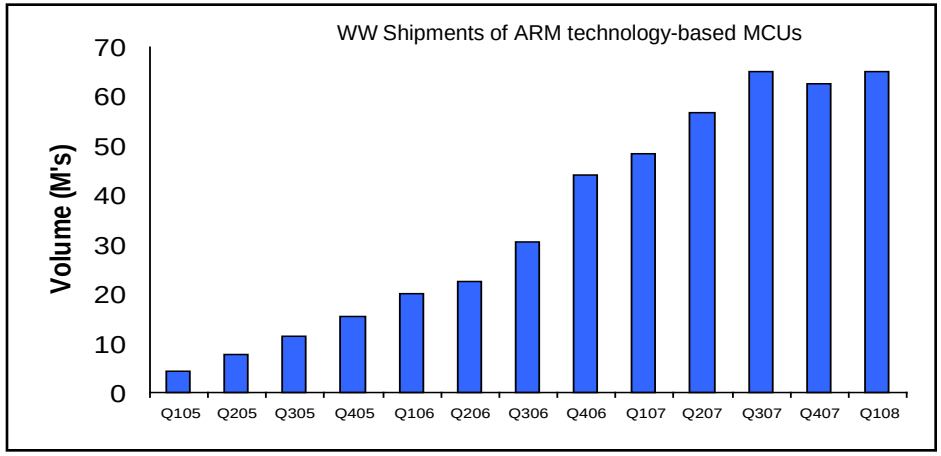
AGENDA

- **About NXP, ARM controllers, and Security Innovation**
- Demystifying Cryptography
- Benefits of software-based encryption
- Dangers of communication via plain text
- Encrypting communications when reading data or code from off-chip memory
- NXP encryption libraries – implementation details and tips

Why ARM?

- 32-bit market is rapidly outgrowing the 8/16-bit markets
- ARM-based MCUs are rapidly outgrowing 32-bit market
  - 140% CAGR
  - Expanding ARM connected community for Tool/Software support





Why NXP's ARM Microcontrollers?

- ARM vendors start with the same ARM IP
  - Cores, Internal Bus, Interrupt Controllers, etc
  - But the end result is not the same!
- Architectural choices, implementation, processing optimization, and power management make a <u>big difference</u>
  - MCU supplier implementation impacts performance, power consumption and ease of use
- Examples:
  - Flash memory performance
  - Peripheral consistency
  - Integration
  - Debugging capabilities

Broadest ARM Core Portfolio

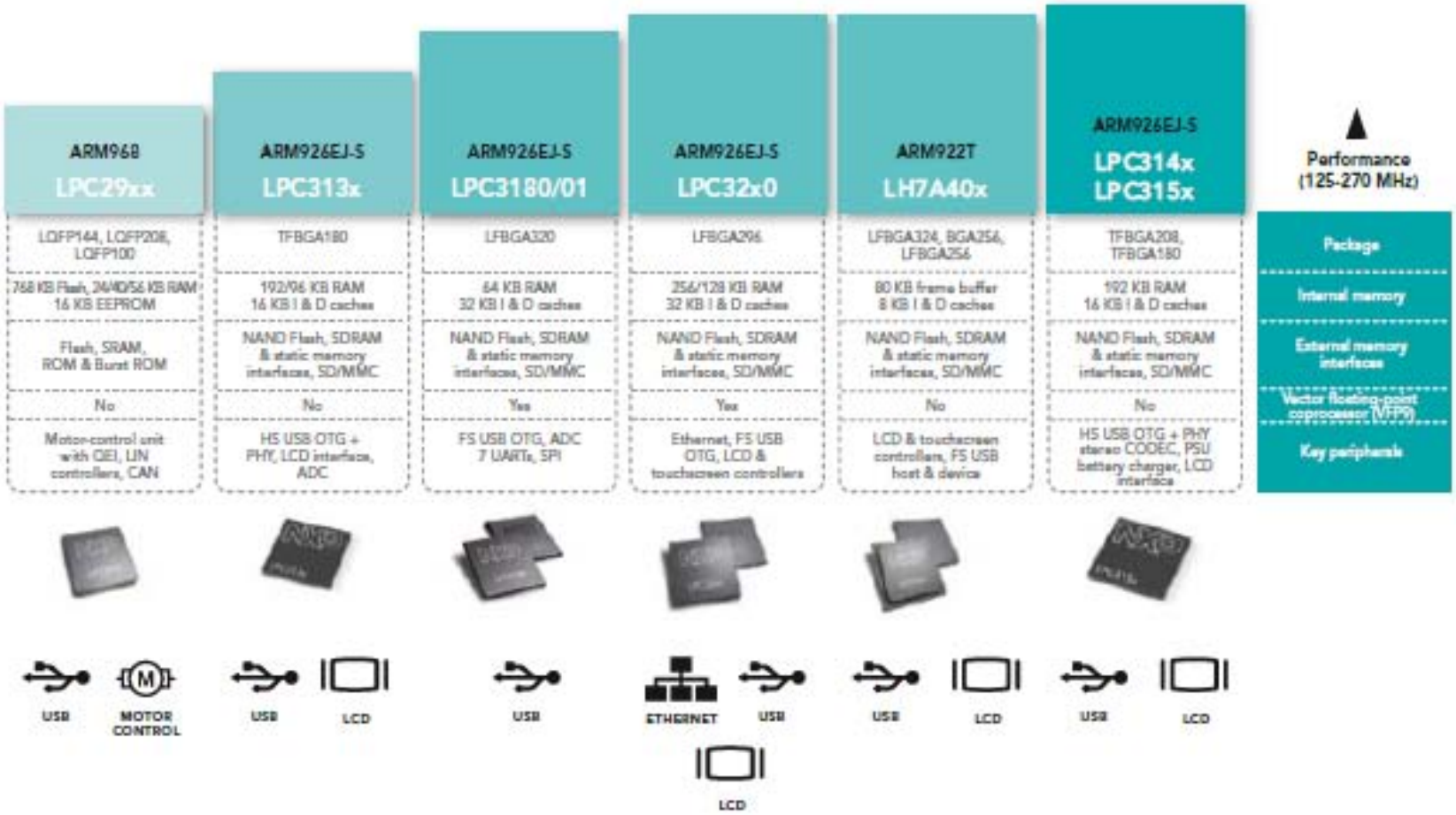
Family Overview

NXP ARM microcontroller continuum

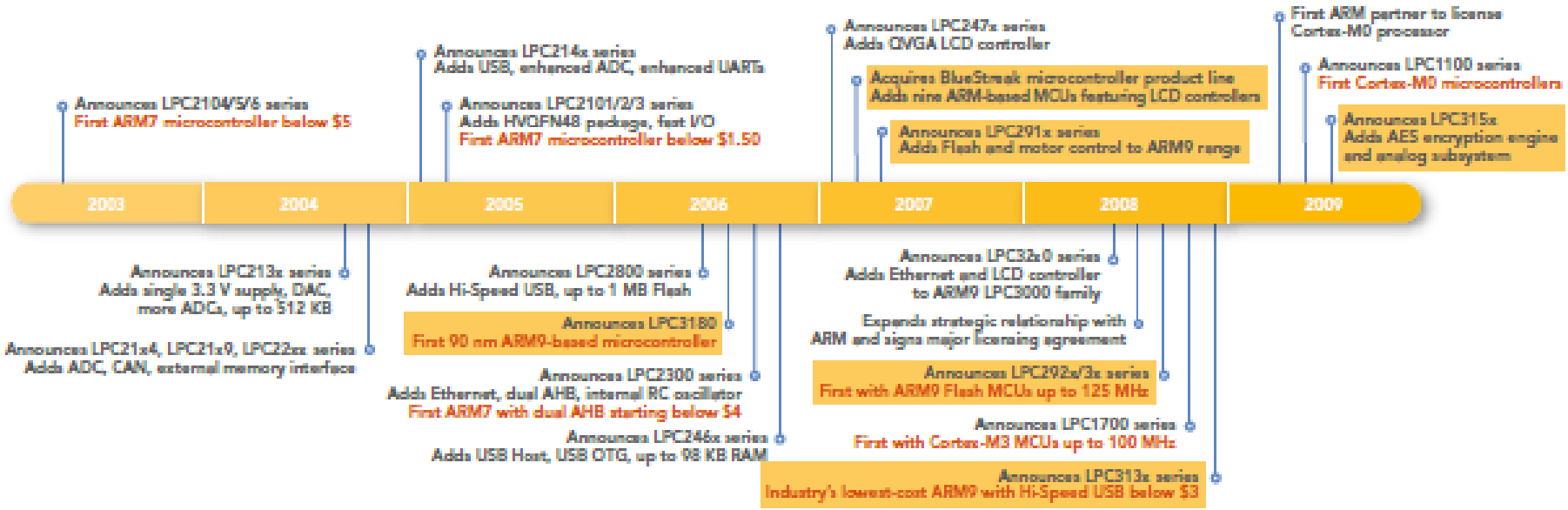
80C51	Cortex	ARM7	ARM9
	Cortex-M0 Cortex-M3	ARMTDMI-S ARM720T (LH7)	ARM926EJ-S ARM922T (LH7A) ARM968 (LPC2900)
LPC900 LPC700	LPC1000	LPC2000	LPC3000
8-bit (Up to 18 MHz)	Low-power/ mixed-signal (Up to 120 MHz)	Mid-range (Up to 84 MHz)	Application processors (Up to 270 MHz)
▸ Industrial ▸ Consumer ▸ Medical ▸ Automotive	▸ Sensors ▸ e-Metering ▸ 16-bit applications ▸ Battery applications	▸ Connectivity ▸ Industrial ▸ Smart control ▸ White goods	▸ Connectivity ▸ Industrial ▸ Consumer ▸ Medical

LPC2000 Family (ARM7)

LPC3000 Family (ARM9)



Changing the MCU Landscape



The Hot 100 products of 2005

Best Product of 2009: Embedded System & IC

LPC213x (2003)

LPC210x (2005)

LPC24xx (2007)

LPC17xx (2009)

Customer Requirements

Some embedded designers desire sophisticated encryption for even low-cost applications that may be carrying or transmitting sensitive information. Key security benefits achieved via an implementation of software encryption schemes are:

- **Confidentiality** - allows the sender and receiver to be sure that the information being shared only in the way they intend
- **Authentication** - allows the receiver of the information to be certain where it came from
- **Integrity** – allows the receiver to verify that the message has not been altered in transit

The NXP & Security Innovation/NTRU Solution

- The NTRU software security libraries for NXP ARM microcontrollers provide the user the tools to **achieve the benefits and features of encryption** and include encryption and decryption of messages, digital signatures, and utilities protocols like key negotiation in an inexpensive and flexible manner
- Encryption algorithms work on the smallest ARM7 LPC2000 through to our ARM926 LPC3000 microcontrollers
- Allows customers who need **short time-to-market** to leverage Ntru's encryption expertise on their products quickly and efficiently
- Customers can **update the software in the field** to keep ahead of hackers and protect their IP

About Security Innovation

- Specialists in application and data security
- Help organizations like EMC, Tyco, Motorola and Sony build more secure software systems
- Acquired NTRU*, a next-generation cryptography firm, in August of 2009.
- NTRU's cryptography is preferred in the telecommunications and embedded markets due to its size, speed and flexibility
- Built strategic partnership with NXP to build encryption libraries specific to the ARM 7/ARM 9 microcontrollers
- Headquartered in Wilmington, MA

* Ntru is short for **N**-th degree ***tru**ncated polynomial ring, or in mathematical notation:* $R[x] / (xN - 1)$

AGENDA

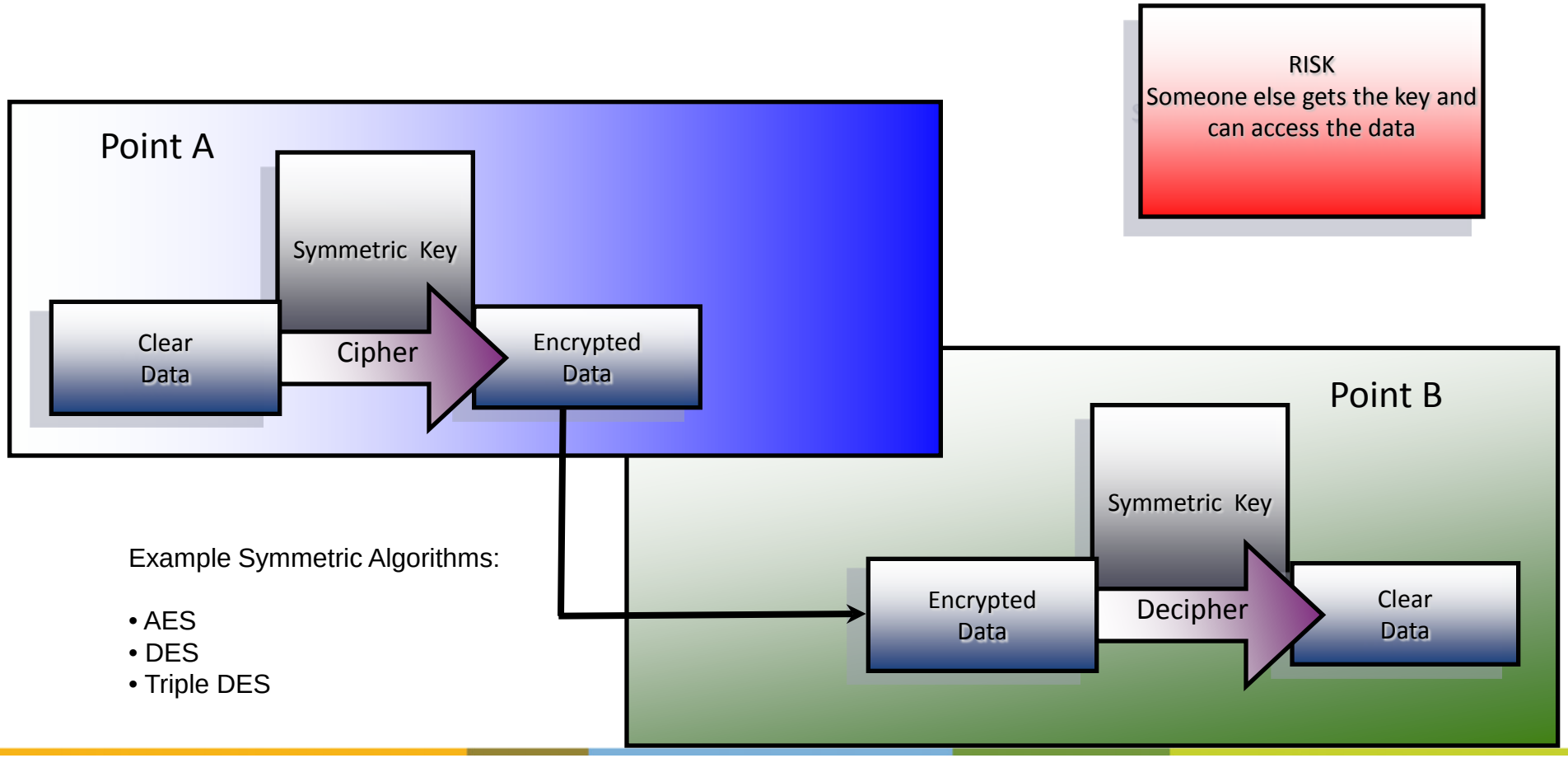
- About NXP, ARM controllers and Security Innovation encryption
- ➢ **Demystifying Cryptography**
- Benefits of software-based encryption
- Dangers of communication via plain text
- Encrypting communications when reading data or code from off-chip memory
- NXP encryption libraries – implementation details and tips

Cryptography Types

- Symmetric Key aka Block Cipher
  - Used to bulk encrypt Data
  - n:n size ratio
  - Same key used to encrypt & decrypt Data
- Asymmetric Key aka Public Key
  - Best used to encrypt secret keys for transmission
  - > n:n ratio
  - Asymmetric key pairs used to encrypt/decrypt
- Digital Signature
  - Used to verify an individual is who they say they are
- Message Digest
  - Used to verify that data is not corrupted or tampered with

Sharing Data - Symmetric Key

Both point A and point B know the secret

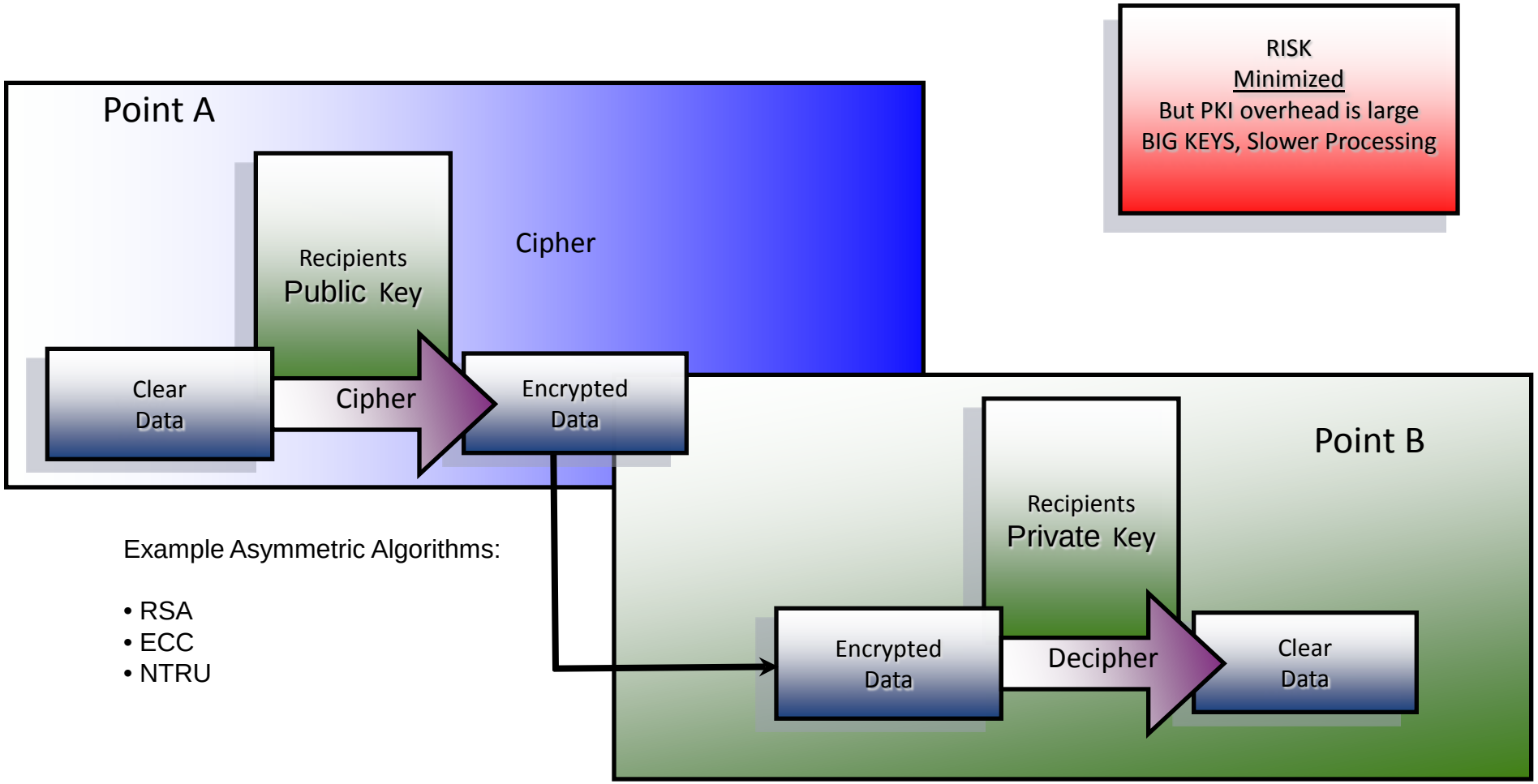


Example Symmetric Algorithms:

- AES
- DES
- Triple DES

Sharing Data – Public Key

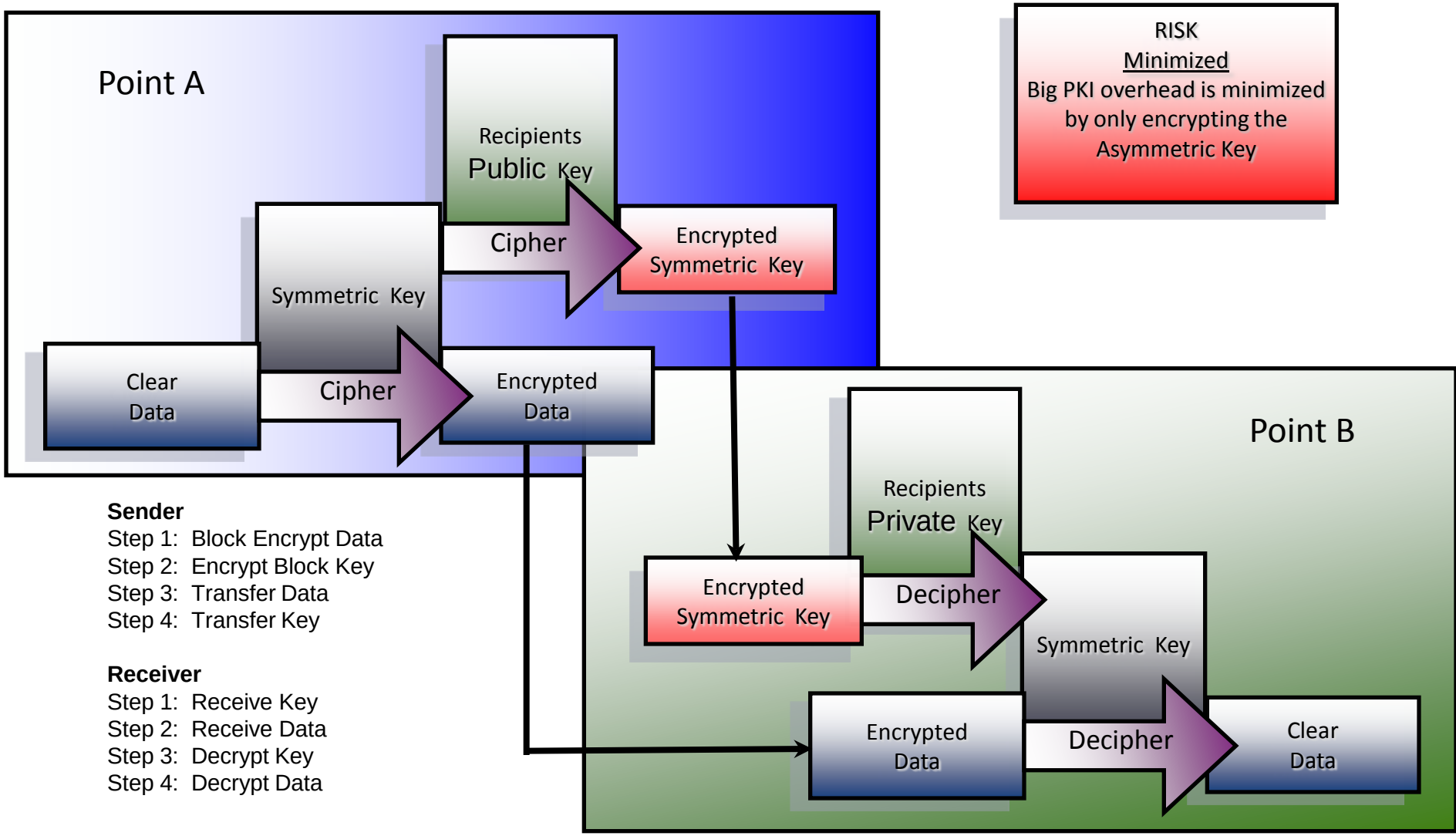
Neither point A nor point B know the complete secret



Example Asymmetric Algorithms:

- RSA
- ECC
- NTRU

Safely Sharing Data

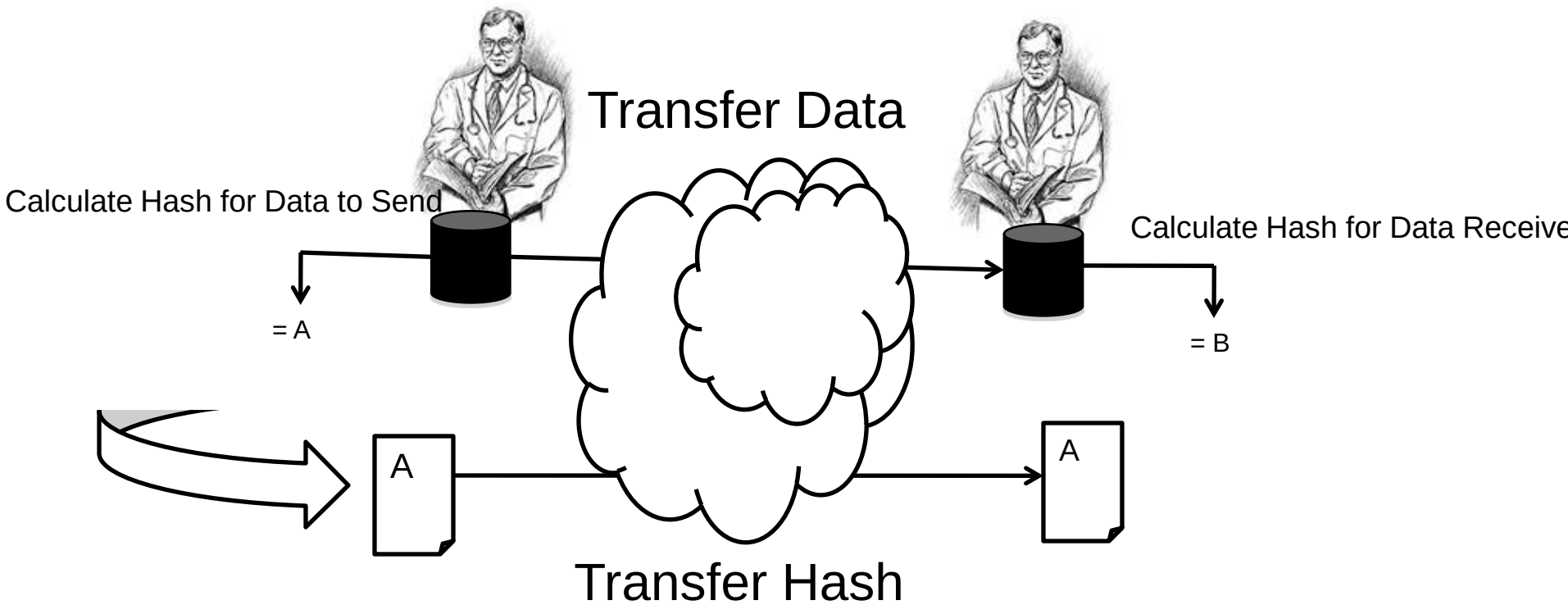


RISK
Minimized
Big PKI overhead is minimized by only encrypting the Asymmetric Key

Sender
Step 1: Block Encrypt Data
Step 2: Encrypt Block Key
Step 3: Transfer Data
Step 4: Transfer Key

Receiver
Step 1: Receive Key
Step 2: Receive Data
Step 3: Decrypt Key
Step 4: Decrypt Data

Hashing



A = B ➔ Data Consistent

A ≠ B ➔ Data Corrupt or Tampered with

AGENDA

- About NXP, ARM controllers, and Security Innovation
- Demystifying Cryptography
- ➢ **Benefits of software-based encryption**
- Dangers of communication via plain text
- Encrypting communications when reading data or code from off-chip memory
- NXP encryption libraries – implementation details and tips

Why Software for Encryption?

- A software implementation of an encryption scheme provides the benefits of **flexibility, speed of implementation, and lower cost** over time.
- Having encryption in software provides the ability to **modify product design and/or product security** without the need to make expensive changes in hardware and the potential resulting changes to the manufacturing process.
- More importantly, the NXP ARM microcontrollers feature **In Application Programming (IAP)** and the popular LPC2300 and LPC2400 series also feature Ethernet, USB and CAN
  - IAP allows customers to periodically change the security algorithm in the field whether or not the product has been comprised
  - Competitive hardware encryption cannot be updated without replacing the microcontroller, which is costly and complicated

Software advantages vs Hardware

- **Cost-effective**: software encryption doesn't require additional circuitry. The use of software encryption shortens design cycles, improves reliability, and lowers deployment costs
- **Battery lifetime**: software encryption runs on the main processor, unlike a hardware coprocessor which draws additional power. Use of software encryption extends battery lifetime.
- **Performance and Flexibility**: software encryption can match the performance of hardware encryption on some other MCUs but by giving customers the flexibility to make changes on installed applications without costly recalls
- **Regulatory**: Government export control rules do not apply until the MCU is programmed with the encryption software

The NTRU software + NXP ARM MCUs offers customers the first general purpose ARM with encryption, Ethernet, USB and other communication peripherals

AGENDA

- About NXP, ARM controllers, and Security Innovation
- Demystifying Cryptography
- Benefits of software-based encryption
- ➢ **Dangers of communication via plain text**
- Encrypting communications when reading data or code from off-chip memory
- NXP encryption libraries – implementation details and tips

Dangers of communication via plain text

It's all about C.I.A. -- Plain text offers none

- **Confidentiality** - allows the sender and receiver to be sure that the information being shared only in the way they intend
- **Integrity** – allows the receiver to verify that the message has not been altered in transit
- **Authentication** - allows the receiver of the information to be certain where it came from

AGENDA

- About NXP, ARM controllers, and Security Innovation
- Demystifying Cryptography
- Benefits of software-based encryption
- Dangers of communication via plain text
- ➢ **Encrypting communications when reading data or code from off-chip memory**
- NXP encryption libraries – implementation details and tips

Requirement when reading data or code

- Encryption and Decryption of Messages
  - allows the sender and receiver to be sure that the information is being shared only in the way they intend
- Encryption and Decryption of Keys
  - allows the sender and receiver to be sure that the secret is being shared only in the way they intend
- Digital signatures
  - allows the receiver to verify that the message has not been altered in transit.

Secure transmission requires multiple activities to ensure the data is securely transferred and that it gets to the proper person intact

AGENDA

- About NXP, ARM controllers, and Security Innovation
- Demystifying Cryptography
- Benefits of software-based encryption
- Dangers of communication via plain text
- Encrypting communications when reading data or code from off-chip memory
- **NXP encryption libraries – implementation details and tips**

Security Innovation NXP Crypto Library

- Security Innovation Crypto for NXP is all hand crafted assembler
  - Provides the highest possible performance
  - Provides the highest possible on chip flexibility
- Small Footprint
  - 2164-6092 bytes depending on function
- High Throughput
  - All functions optimized for speed
- High Reliability
  - Not open source, professionally developed and maintained

Encryption suite components

- Hash algorithms
  - SHA-1
  - MD5
- Symmetric-key encryption/decryption
  - AES 128, 192, 256
  - Triple-DES (64 x 3)
  - Modes (ECB, CBC, CTR, …)
- Asymmetric-key encryption/decryption
  - RSA 1024, 2048
  - Diffie-Hellman 1024, 2048
- Digital Signature
  - RSA, DSA
- Random Number Generator
  - X9.82

Library is granular, customers can choose the encryption components that best fit their application's requirements

Flash sizes for each Library

Library	Bytes
SHA-1	2164
MD5	2756
PRNG	2888
TDES Core	4176
AES Core	3448
DH	5128
DSA	7776
RSA Encrypt/Decrypt	6092
RSA Sign/Verify	3224

Throughputs

Hash Algorithm	
Name	**Throughput (kbytes /sec)**
SHA-1	1915
MD5	3516
Symmetric algorithms	
Name	**Throughput (kbytes /sec)**
AES-CBC	825
AES-ECB	874
AES-CCM, CT only	373
AES-CCM, AD only	816
3DES-CBC	326
3DES-CTR	317
3DES-ECB	333

CT = CipherText (encrypt + authenticate)
AD = Associated Data (authenticate only)

Throughputs (continued)

Asymmetric, Encrypt/Decrypt	
Name	Time (sec)
RSA-1024 encrypt	0.01
RSA-1024 decrypt	0.27
RSA-2048 encrypt	0.05
RSA-2048 decrypt	2.13

Asymmetric, Sign/Verify	
Name	Time (sec)
RSA-1024 sign	0.27
RSA-1024 verify	0.01
RSA-2048 sign	2.13
RSA-2048 verify	0.05
DSA-1024 sign	0.17
DSA-1024 verify	0.33

Diffie-Hellman		
Modulus Size (bits)	Private Key size (bits)	Time (sec)
1024	160	0.17
1024	1024	1.08
2048	224	0.93
2048	2048	8.48

Example application structure – encryption

SENDER:

- Allocate memory buffers
- Create and seed Random Byte Generator
- Generate AES-CCM key
- Generate AES-CCM nonce
- Set AES-CCM MAC length (typically 8 or 16 bytes)
- Obtain receiver's RSA public key
  - Storage & transmission of public keys is managed externally to the suite
- Encrypt AES-CCM key with receiver's RSA public key
- Encrypt and authenticate message using:
  - AES-CCM key
  - AES-CCM Nonce
  - AES-CCM MAC length
- Send
  - RSA ciphertext
    - This is the encrypted AES-CCM key
  - AES-CCM nonce
  - AES-CCM MAC length
  - AES-CCM ciphertext + length
    - This is the encrypted message

RECEIVER:

- Allocate memory buffers
- Receive
  - RSA ciphertext
    - This is the encrypted AES-CCM key
  - AES-CCM nonce
  - AES-CCM MAC length
  - AES-CCM ciphertext + length
    - This is the encrypted message
- Use receiver's RSA private key to decrypt RSA ciphertext and recover AES-CCM key
  - Reject message if this fails
  - Storage of RSA private key is managed externally to the suite
- Decrypt and check authentication on message
  - Use AES-CCM with inputs:
    - AES-CCM nonce
    - AES-CCM MAC length
    - AES-CCM ciphertext + length
  - Reject message if this does not return success
- Output decrypted message
- No need to use Random Byte Generator

Example application structure – signing

SENDER:

- Allocate memory buffers
- Hash message with SHA-1
- Use RSA private key to sign hash
  - Storage of RSA private key is managed externally to the suite
- Send
  - Message
  - Signature
- No need to use Random Byte Generator

RECEIVER:

- Allocate memory buffers
- Receive
  - Message
  - Signature
- Obtain sender's RSA public key
  - Storage & transmission of public keys is managed externally to the suite
- Hash message with SHA-1
- Verify signature using RSA
  - Inputs are:
    - Message
    - Signature
    - RSA public key
  - Reject message if this does not return success
- Output message
- No need to use Random Byte Generator

Sample RSA-1024 Encrypt/Decrypt

```
/*********************************************************
 * Test valid RSA-1024 encryption and decryption *
 *********************************************************/

 /* Keys */
  uint8_t const n_1024[] = { /* … */ };
  uint8_t const e[] = { /* … */ };
  uint8_t const sk_1024[] = { /* … */ };

 /* Data */
 uint8_t const m[] = { /* … */ };
  uint8_t dec_m[128-11];
 uint32_t dec_m_byte_len = sizeof(dec_m);
 uint8_t  c[128];
 uint32_t e_working_buf[215];
 uint32_t d_working_buf[96];
 uint32_t retcode;

 /* seed random number generator */
 if ((retcode = nacl_drbg_seed(e_working_buf, seed)) != NACL_SUCCESS) {
       /* handle error */
 }

 /* encrypt a short message */
 if ((retcode = nacl_rsa_encrypt(m, sizeof(m), n_1024, 128, e, sizeof(e), e_working_buf, c))
                                        != NACL_SUCCESS) {
       /* handle error */
 }

 /* decrypt the ciphertext */
  if ((retcode = nacl_rsa_decrypt(c, 128, sk_1024, d_working_buf, dec_m,
                                        &dec_m_byte_len)) != NACL_SUCCESS) {
       /* handle error */
 }

 /* make sure the message retrieved is equal to the original one */
  check_plaintext(dec_m, dec_m_byte_len, m, sizeof(m));
```

Notes

- All functions return an error code (success == NACL_SUCCESS = 0)
- Public encryption key is the pair (n_1024, e)
- Private decryption key is the structure sk_1024
- check_plaintext is essentially a friendly name for memcmp
- All memory is allocated externally to the library
  - c: ciphertext buffer (size depends on RSA key size)
  - dec_m: output decrypted message buffer (size depends on RSA key size and padding size)
  - e_working_buf, d_working_buf: scratch memory used by libraries
- Random byte generator must be seeded – creation of the seed from a random source is external to the suite

Sample RSA-2048 Sign/Verify

```
/*****************************************************
 * Test valid RSA-2048 signing and verification *
 *****************************************************/

/* Keys */
 uint8_t const n_2048[] = { /* … */ };
 uint8_t const e[] = { /* … */ };
 uint8_t const sk_2048[] = { /* … */ };

/* Data */
 uint8_t const h[] = { /* … */ };
 uint8_t  s[256];
 uint32_t s_working_buf[192];
 uint32_t v_working_buf[256];
 uint32_t retcode;

/* sign a message hash */
 if ((retcode = nacl_rsa_sign(256, h, sizeof(h), sha1_asn_id, sizeof(sha1_asn_id),
                              sk_2048, s_working_buf, s)) != NACL_SUCCESS) {
      /* Handle error */
}

/* verify the (message hash, signature) pair */
 if ((retcode = nacl_rsa_verify(s, h, sizeof(h), sha1_asn_id, sizeof(sha1_asn_id), n_2048,
                                256, e, sizeof(e), v_working_buf)) != NACL_SUCCESS) {
   /* Handle error */
}

/* flip a byte of the signature, and attempt to verify it */
  s[10] ^= 0xff;
  if ((retcode = nacl_rsa_verify(s, h, sizeof(h) sha1_asn_id, sizeof(sha1_asn_id), n_2048, 256,
                                 e, sizeof(e), v_working_buf)) != NACL_INVALID_SIGNATURE) {
   /* Handle error */
}
```

Notes

- All functions return an error code (success == NACL_SUCCESS = 0)
- Random Byte Generator need not be seeded
- Sign takes:
  - Message hash
  - Private signing key (the structure sk_2048))
  - Public modulus, n_2048
  - ID for the hash function
- Verify takes:
  - Signature
  - Message hash
  - Public verification key (the pair (n_2048, e))
- If message, signature, or verification key has been altered, verify returns NACL_INVALID_SIGNATURE
- All memory is allocated externally to the library
  - s: signature buffer (size depends on RSA key size)
  - s_working_buf, v_working_buf : scratch memory used by libraries

Sample AES-CCM Encrypt/Decrypt

```
/*********************************************************
 * Test valid AES-CCM encryption and decryption *
 *********************************************************/

  /* Keys */
 uint8_t const key[16] = { /* … */ };
 uint32_t enc_rk[44];
 uint32_t dec_rk[44];


 /* Data */
 uint8_t const nonce[]= { /* … */ };
 uint8_t const adata[] = { /* … */ };
 uint8_t const pt[] = { /* … */ };
 uint8_t dec_pt[sizeof(pt)];
 uint32_t n_mac_bytes = 16;
 uint8_t ct[sizeof(pt) + n_mac_bytes];
 uint32_t working_buf[92];


 /* create round keys */
 if ((retcode = nacl_aes128_enc_key_schedule(key, enc_rk)) != NACL_SUCCESS) {
        /* handle error */
 }
 if ((retcode = nacl_aes128_dec_key_schedule(enc_rk, dec_rk)) != NACL_SUCCESS) {
        /* handle error */
 }


/* encrypt the plaintext */
 if ((retcode = nacl_aes128_ccm_encrypt(working_buf, nonce, sizeof(nonce), adata,
                    sizeof(adata), pt, sizeof(pt), n_mac_bytes, enc_rk, ct)) != NACL_SUCCESS) {
        /* handle error */
 }


/* decrypt the ciphertext */
 if ((retcode = nacl_aes128_ccm_decrypt(working_buf, nonce, sizeof(nonce), adata,
                    sizeof(adata), ct, sizeof(ct), n_mac_bytes, enc_rk, pt)) != NACL_SUCCESS) {
        /* handle error */
 }


/* make sure the decrypted plaintext is equal to the original */
 check_plaintext(dec_pt, sizeof(ct) – n_mac_bytes,  pt, sizeof(pt));
```

Notes

- All functions return an error code (success == NACL_SUCCESS = 0)
- Random Byte Generator need not be seeded
- Generate encryption and decryption round keys from the AES key:
  - Storing round keys improves performance for subsequent use of an AES key, or for faster switching between AES keys
- Encrypt takes:
  - working_buf: scratch memory used by the libraries
  - nonce: used to randomize starting point of counter-mode encryption. Must be unique to this call.
  - adata: data that will be authenticated but not encrypted
  - pt: plaintext
  - n_mac_bytes: length of authentication code
  - enc_rk: encryption round keys derived from the AES key
  - ct: buffer where ciphertext will be output: must be pre-allocated and large enough for ciphertext (plaintext size + MAC)
- Decrypt takes:
  - working_buf: scratch memory used by the libraries
  - nonce: must be the same as the one used by the encryptor
  - adata: must be the same as that used by the encryptor
  - ct: ciphertext for decryption
  - n_mac_bytes: length of authentication code
  - dec_rk: decryption round keys derived from the encryption round keys
  - dec_pt: buffer where plaintext will be output: must be pre-allocated and large enough for plaintext (ciphertext size – MAC)

Conclusion

- Security Innovation NXP Libraries:
  - Are hand crafted in assembler specifically for the chip and provide the highest possible performance in software
  - Are flexible and allow you to update software in the field to keep ahead of hackers and protect your IP
  - Have an extremely small footprint for effectiveness and efficiency
  - Provide high quality documentation for rapid implementation
  - Are professionally developed and maintained for maximum reliability
- Provides all of the key cryptographic support for high strength systems
  - **Asymmetric/Public Key**
  - **Symmetric/Block**
  - **Digital Signatures**
  - **Message Digests**

Questions?

- **Cost**:
  - One-time development kit cost is $1,500/seat
  - Low “per unit” license fee
- **Support**
  - free 30-day evaluation
  - detailed user’s guide
- **Contact**
  - Pete Jenney, +1.978.694.1008 x30, pjenney@securityinnovation.com
  - For NXP ARM questions, gene.carter@nxp.com
- **Support for Cortex pending**
- **Partnership with PolarSSL pending**