

# Using BSAFE

Steve Burnett  
RSA Data Security, Inc.  
[burnetts@rsa.com](mailto:burnetts@rsa.com)

RSA Data Security Conference  
January 30, 1997



# Topics

- 1. What is BSAFE**
- 2. The BSAFE Model**
- 3. Variation On The Theme -- Random Algorithm**
- 4. Variation On The Theme -- RSA Keypair Generation**
- 5. Subroutine Calls**
- 6. Actual Code**
- 7. The Algorithm Chooser**
- 8. The Surrender Context**
- 9. Security Considerations**



# BSAFE

- General Purpose Cryptography Toolkit
- Written in C
- Portable
- Object-Oriented
  - » flexible
  - » extensible



# BSAFE

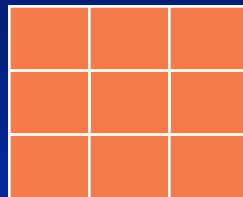
- Executes the most popular cryptographic algorithms: RSA, DSA, DH, DES, RC series, MD2, MD5, SHA-1, Bloom-Shamir.
- BER encodes algorithm and key identifiers.
- Produces digital signatures following industry standards.



# The BSAFE Model



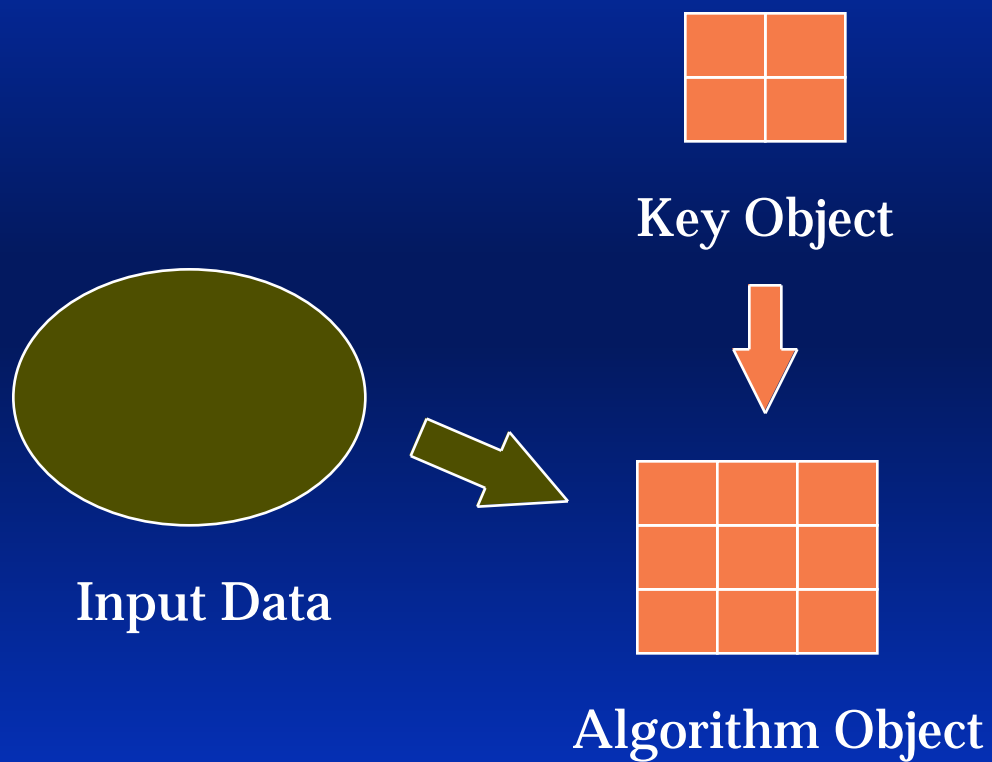
Key Object



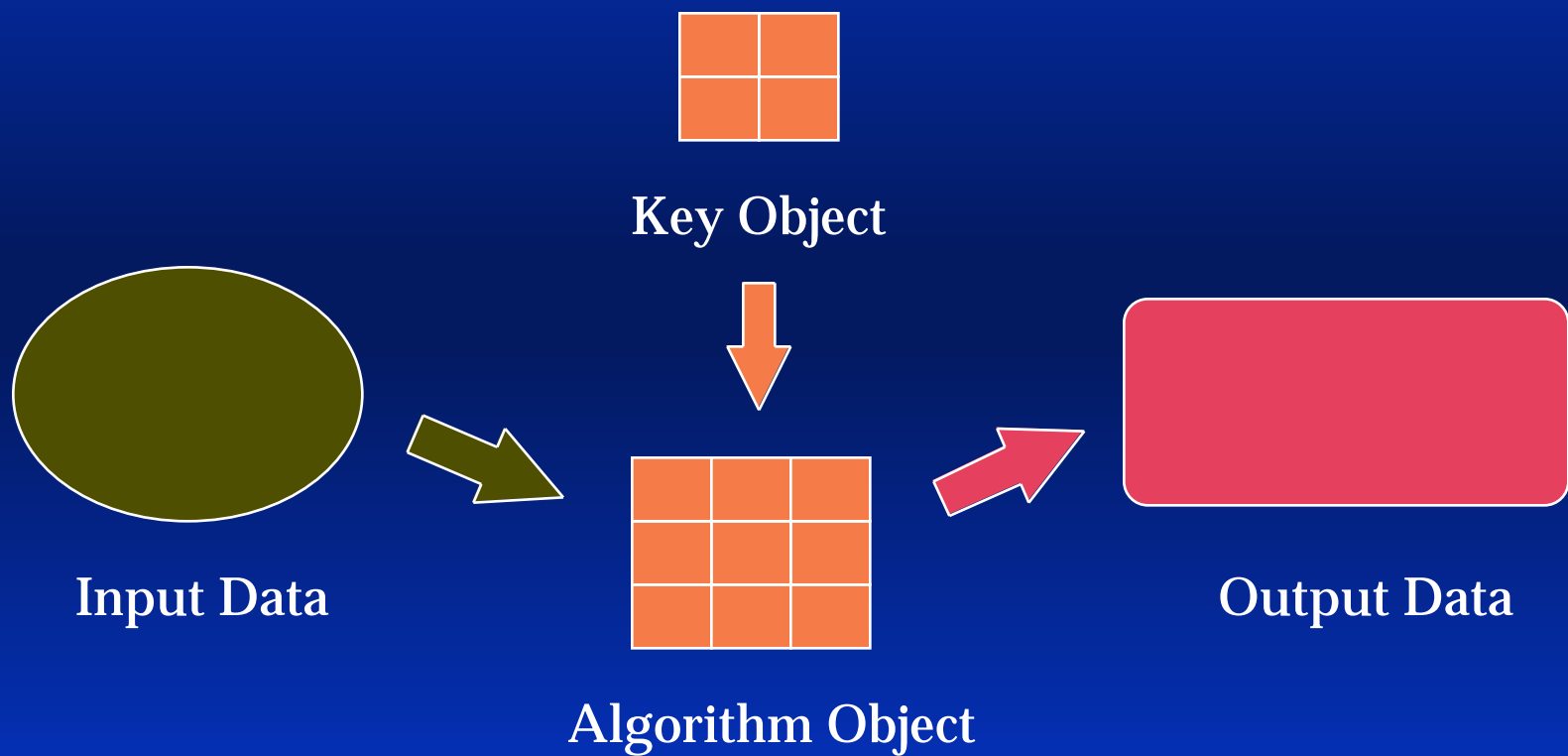
Algorithm Object



# The BSAFE Model



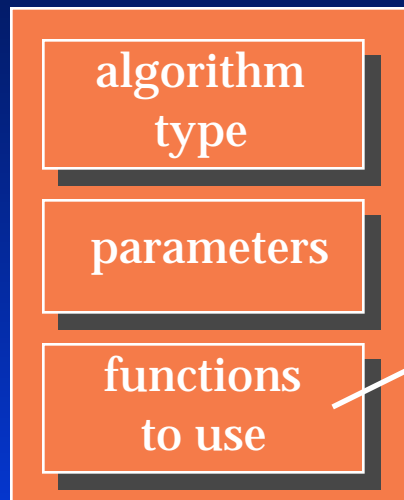
# The BSAFE Model



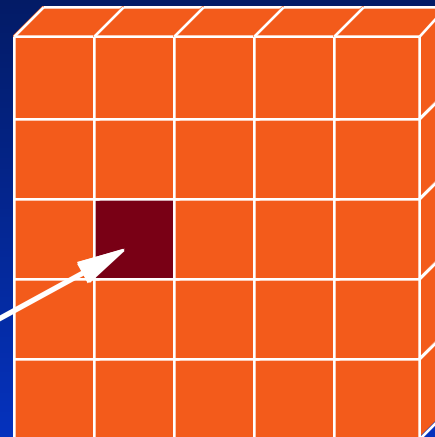
# Definition

An **Algorithm Object** is the “vessel” that carries the information necessary to perform a desired operation.

algorithm object



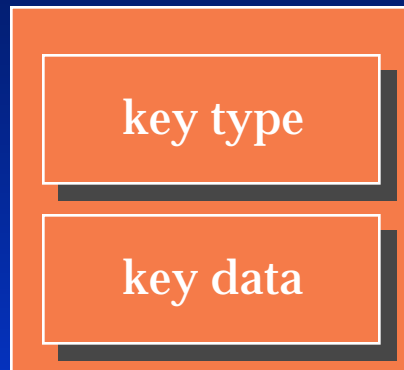
BSAFE Library



# Definition

A **Key Object** is the “vessel” that carries the information about the key employed by the algorithm object.

key object



# Six Steps

---

1. Create
2. Set
3. Init
4. Update
5. Final
6. Destroy



# Step 0: Include Files

---

```
#include "aglobal.h"  
#include "bsafe.h"  
#include "demochos.h"
```



# Step 1: Create

## B\_CreateAlgorithmObject (&rc5Encrypter)

### Algorithm Object

Name: rc5Encrypter

Type:

Special Info:

Key Object:

Chooser:



# Step 2: Set

## B\_SetAlgorithmInfo

### Algorithm Object

Name: rc5Encrypter

Type: AI\_RC5\_CBC

Special Info:

Key Object:

Chooser:

version  
rounds  
word size  
initialization vector



# Step 3: Init

## B\_EncryptInit

### Algorithm Object

Name: rc5Encrypter

Type: AI\_RC5\_CBC

Special Info: v, r, w, iv

Key Object:

Chooser: DEMO\_

### Key Object

Name: ...

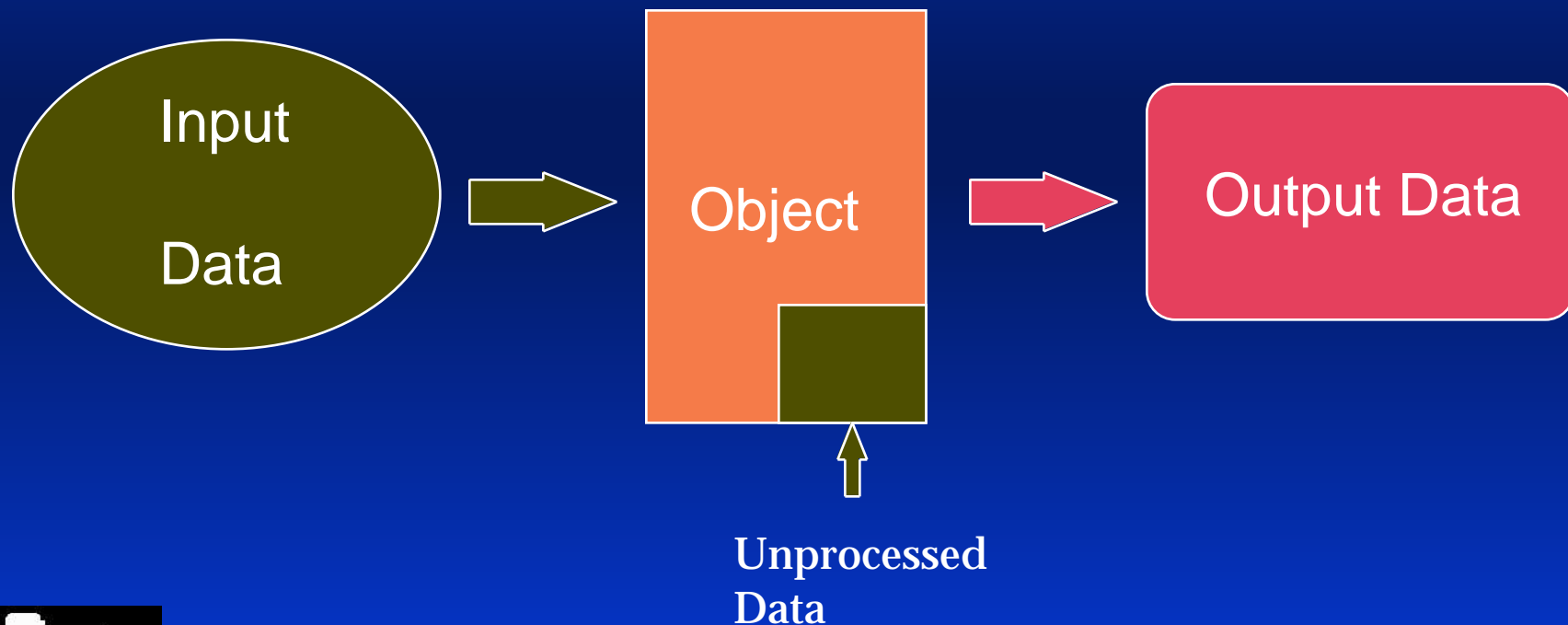
Type: ...

Data: ...



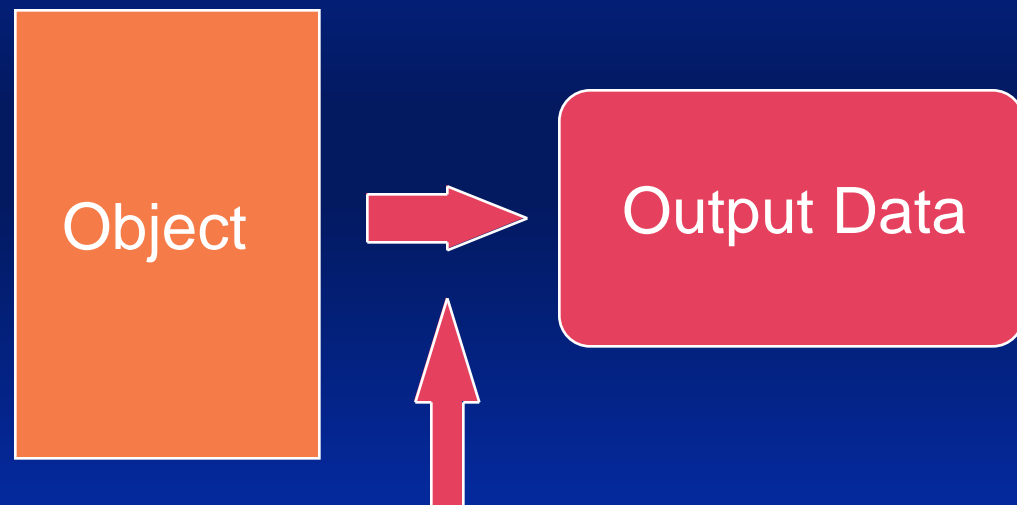
# Step 4: Update

## B\_EncryptUpdate



# Step 5: Final

## B\_EncryptFinal



Previously  
Unprocessed  
Data



# Step 3: Init

## B\_EncryptInit

### Algorithm Object

Name: rc5Encrypter

Type: AI\_RC5\_CBC

Special Info: v, r, w, iv

Key Object:

Chooser: DEMO\_

### Key Object

Name: ...

Type: ...

Data: ...



# Step 3a: Create A Key Object

**B\_CreateKeyObject (&rc5Key)**

Key Object

Name: rc5Key

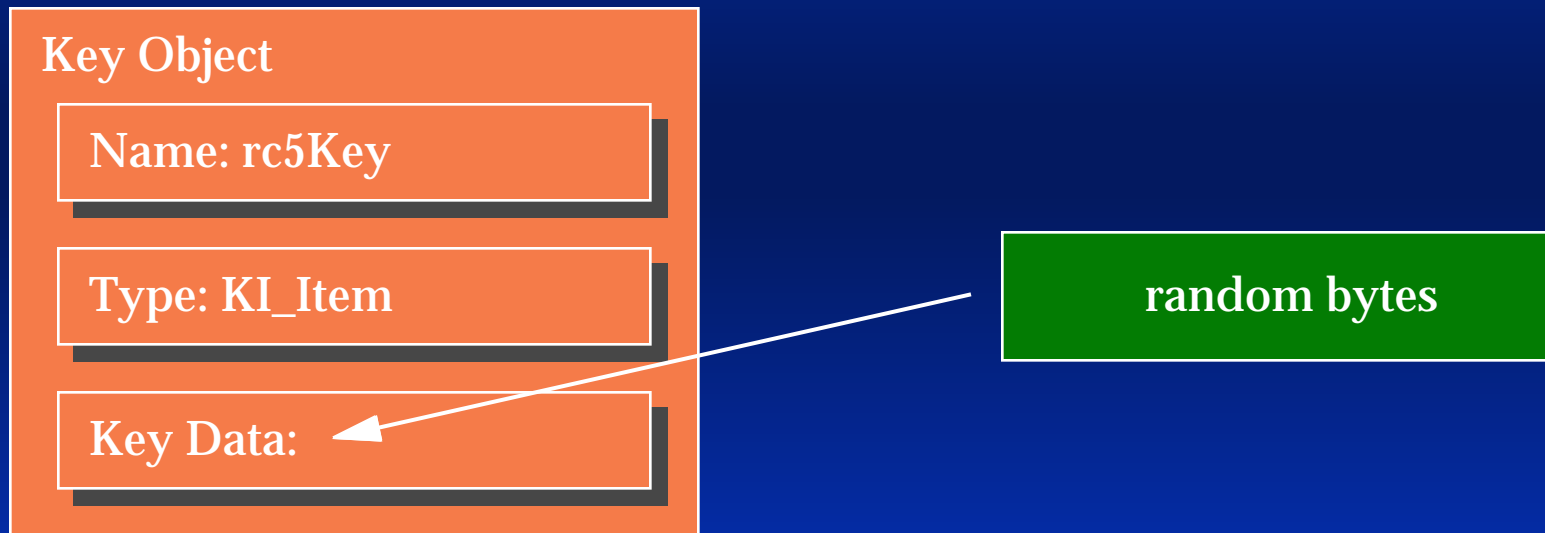
Type:

Key Data:



# Step 3b: Set The Key Object

## B\_SetKeyInfo



# Current State

## Algorithm Object

Name: rc5Encrypter

Type: AI\_RC5\_CBC

Special Info: v, r, w, iv

Key Object:

Chooser: DEMO\_

## Key Object

Name: rc5Key

Type: KI\_Item

Data: <random bytes>



# Step 6: Destroy Algorithm Object

## B\_DestroyAlgorithmObject



### Key Object

Name: rc5Key

Type: KI\_Item

Data: <random bytes>



# Step 6: Destroy Key Object

**B\_DestroyKeyObject**



# Variation On The Theme

---

## Random Number Generation



Copyright 1997 RSA Data Security, Inc. All rights reserved.

# Step 1: Create

## B\_CreateAlgorithmObject (&random)

### Algorithm Object

Name: random

Type:

Special Info:

Key Object:

Chooser:



# Step 2: Set

## B\_SetAlgorithmInfo

### Algorithm Object

Name: random

Type: AI\_MD5Random

Special Info:

Key Object:

Chooser:



# Step 3: Init

## B\_RandomInit

### Algorithm Object

Name: random

Type: AI\_MD5Random

Special Info:

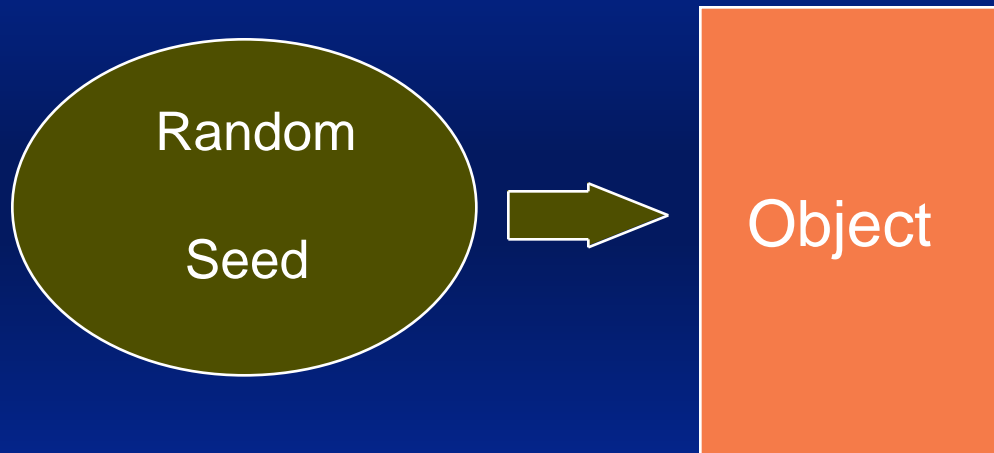
Key Object:

Chooser: DEMO\_



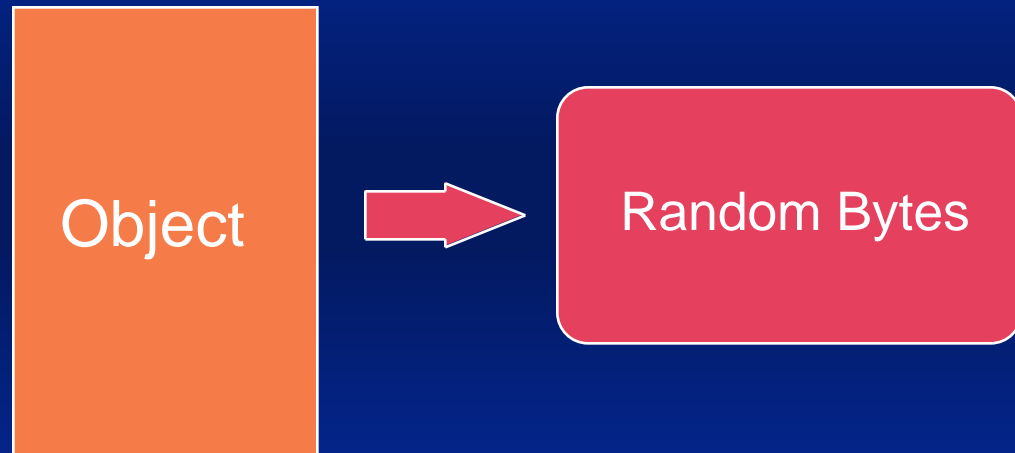
# Step 4: Update

**B\_RandomUpdate**



# Step 5: Generate

**B\_GenerateRandomBytes**



# Step 6: Destroy

## B\_DestroyAlgorithmObject



# Variation On The Theme

---

## RSA Keypair Generation



# Step 3a: Create Two Key Objects

**B\_CreateKeyObject (&rsaPublicKey)**

**B\_CreateKeyObject (&rsaPrivateKey)**

Key Object

Name: rsaPublicKey

Type:

Key Data:

Key Object

Name: rsaPrivateKey

Type:

Key Data:



# Step 1: Create Algorithm Object

## B\_CreateAlgorithmObject (&rsaKeyGenerator)

Algorithm Object

Name: rsaKeyGenerator

Type:

Special Info:

Key Object:

Chooser:



# Step 2: Set

## B\_SetAlgorithmInfo

### Algorithm Object

Name: rc5Encrypter

Type: AI\_RSAKeyGen

Special Info:

Key Object:

Chooser:

modulus bits  
public exponent



# Step 3: Init

## B\_GenerateInit

### Algorithm Object

Name: rc5Encrypter

Type: AI\_RSAKeyGen

Special Info: bits, exp

Key Object:

Chooser: DEMO\_



# Step 4:

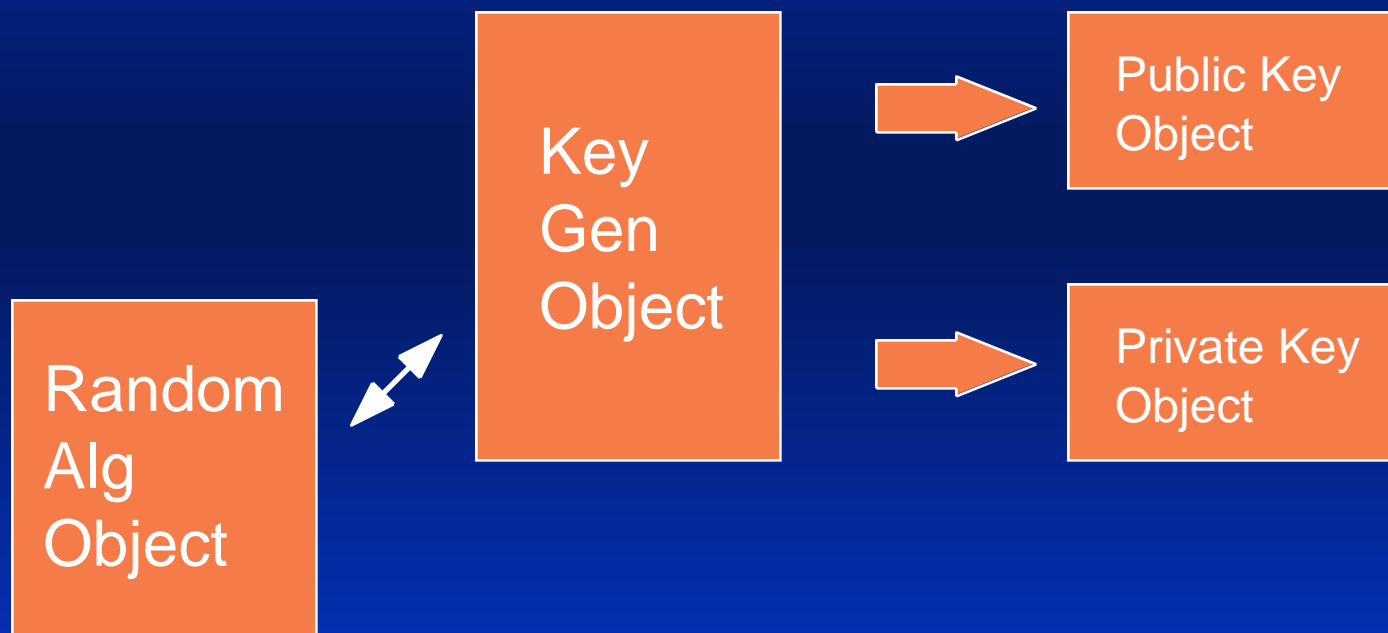
---

No step 4 when generating an RSA keypair



# Step 5: Generate Keypair

## B\_GenerateKeypair



# Generated Key Objects

## Key Object

Name: rsaPublicKey

Type: KI\_RSAPublic

Key Data: 01 00 01 ...

## Key Object

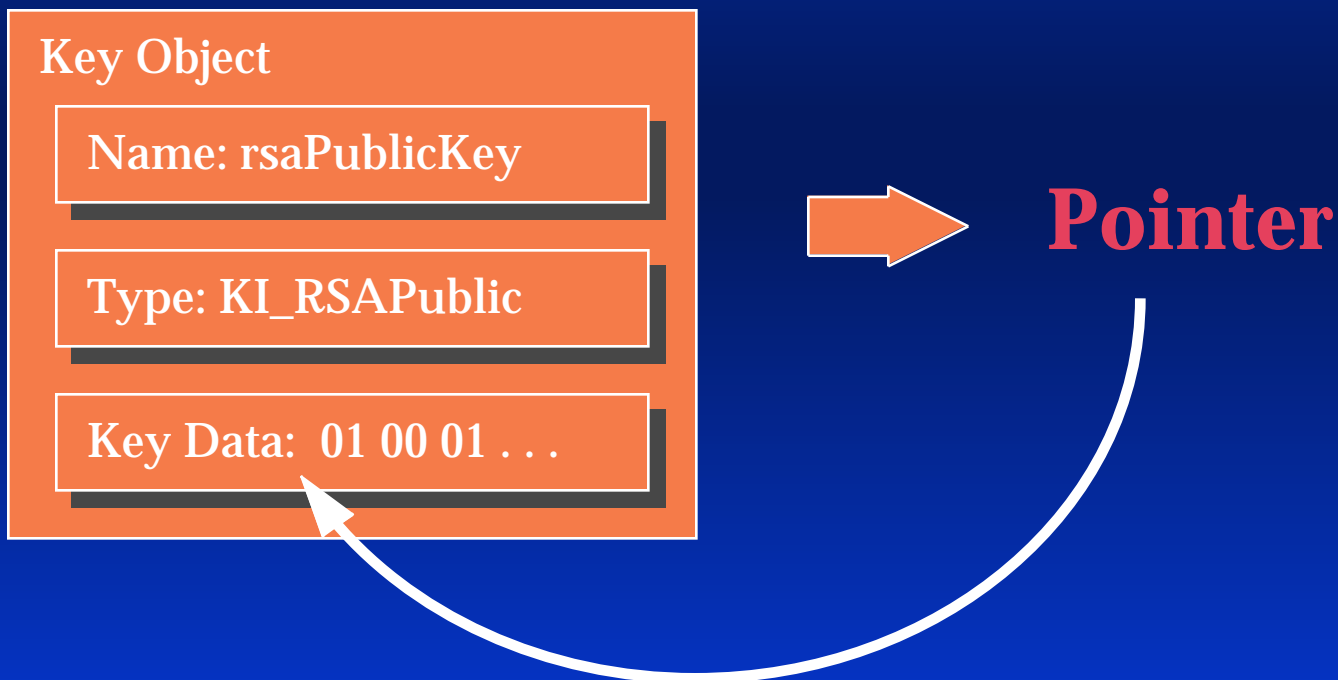
Name: rsaPrivateKey

Type: KI\_RSAPrivate

Key Data: c4 50 2f ...



# B\_GetKeyInfo



# B\_GetKeyInfo

```
A_RSAPublic *publicKeyInfo;
```

```
if ((status = B_GetKeyInfo  
    ((POINTER *)&publicKeyInfo, rsaPublicKey,  
    KI_RSAPublic)) != 0)  
    break;
```

**Pointer**



```
publicKeyInfo->modulus.data = b1 39 . . .  
publicKeyInfo->modulus.len = 96  
publicKeyInfo->exponent.data = 01 00 01  
publicKeyInfo->exponent.len = 3
```



# B\_GetKeyInfo

```
ITEM *publicKeyInfoBER;
```

```
if ((status = B_GetKeyInfo  
    ((POINTER *)&publicKeyInfoBER, rsaPublicKey,  
    KI_RSAPublicBER)) != 0)  
    break;
```

**Pointer**



```
publicKeyInfoBER->data = 30 7c 30 0d . . .  
publicKeyInfoBER->len = 126
```



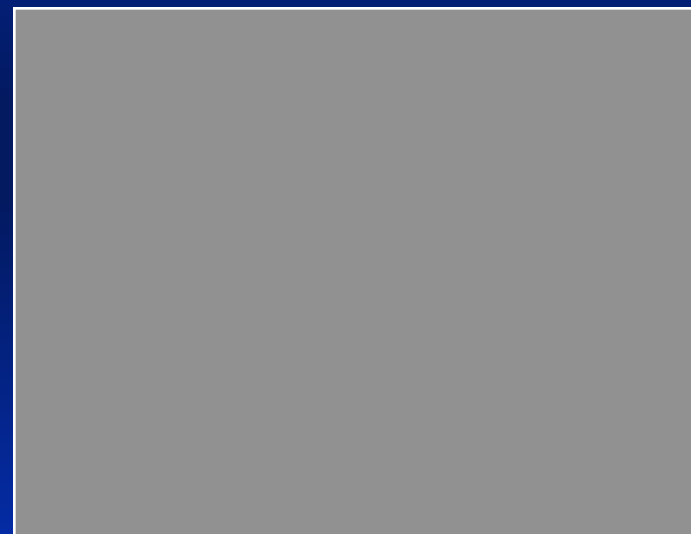
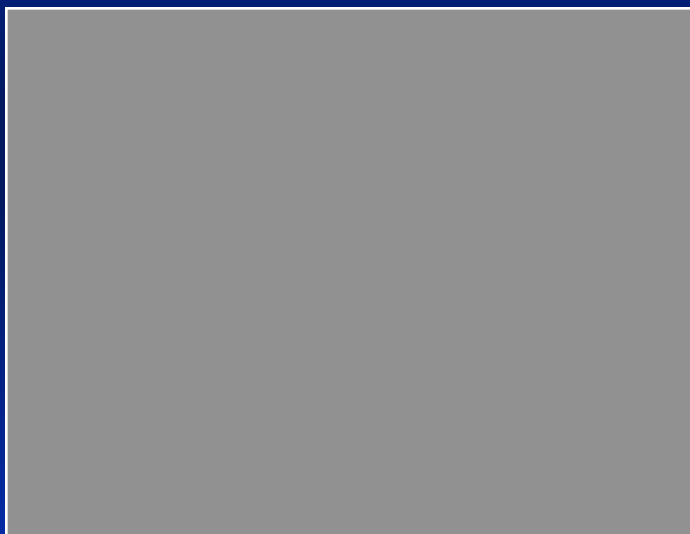
# Step 6: Destroy Algorithm Object

**B\_DestroyAlgorithmObject**



# Step 6: Destroy Key Objects

**B\_DestroyKeyObject**



# Subroutine Calls -- Create

---

B\_CreateAlgorithmObject

B\_CreateKeyObject



# Subroutine Calls -- Set

---

B\_SetAlgorithmInfo

B\_SetKeyInfo



# Subroutine Calls -- Init, Update, Final

B\_EncryptInit

B\_EncryptUpdate

B\_EncryptFinal

B\_DecryptInit

B\_DecryptUpdate

B\_DecryptFinal

B\_DigestInit

B\_DigestUpdate

B\_DigestFinal

B\_EncodeInit

B\_EncodeUpdate

B\_EncodeFinal

B\_DecodeInit

B\_DecodeUpdate

B\_DecodeFinal



# Subroutine Calls -- Init, Update, Final

B\_SignInit

B\_SignUpdate

B\_SignFinal

B\_VerifyInit

B\_VerifyUpdate

B\_VerifyFinal

B\_RandomInit

B\_RandomUpdate

B\_GenerateRandomBytes



# Subroutine Calls -- Init, Update, Final

B\_GenerateInit

B\_GenerateKeypair

B\_GenerateInit

B\_GenerateParameters

B\_KeyAgreeInit

B\_KeyAgreePhase1

B\_KeyAgreePhase2



# Subroutine Calls -- Destroy

---

`B_DestroyAlgorithmObject`

`B_DestroyKeyObject`



# Example

---

**Write a program that will prompt the user for data to encrypt. The program should then encrypt the input and display the encrypted data.**



# Algorithm Choosers

```
B_ALGORITHM_METHOD *SAMPLE_CHOOSER[ ] = {  
    &AM_RC5_CBC_ENCRYPT,  
    &AM_RC5_CBC_DECRYPT,  
    (B_ALGORITHM_METHOD *)NULL_PTR  
};
```



# Algorithm Choosers

in mychoosr.h

```
extern B_ALGORITHM_METHOD *SAMPLE_CHOOSER[ ];
```

in source code

```
include "mychoosr.h"
```



# Algorithm Choosers

in mychoosr.c

```
#include "mychoosr.h"
```

```
B_ALGORITHM_METHOD *SAMPLE_CHOOSER[ ] = {  
    &AM_RC5_CBC_ENCRYPT,  
    &AM_RC5_CBC_DECRYPT,  
    (B_ALGORITHM_METHOD *)NULL_PTR  
};
```



# Algorithm Choosers

```
/* An algorithm chooser for RSA operations. */

#include "mychoosr.h"

B_ALGORITHM_METHOD *SAMPLE_CHOOSER[ ] = {
    &AM_RSA_CRT_ENCRYPT,
    &AM_RSA_DECRYPT,
    (B_ALGORITHM_METHOD *)NULL_PTR
};
```



# Surrender Context

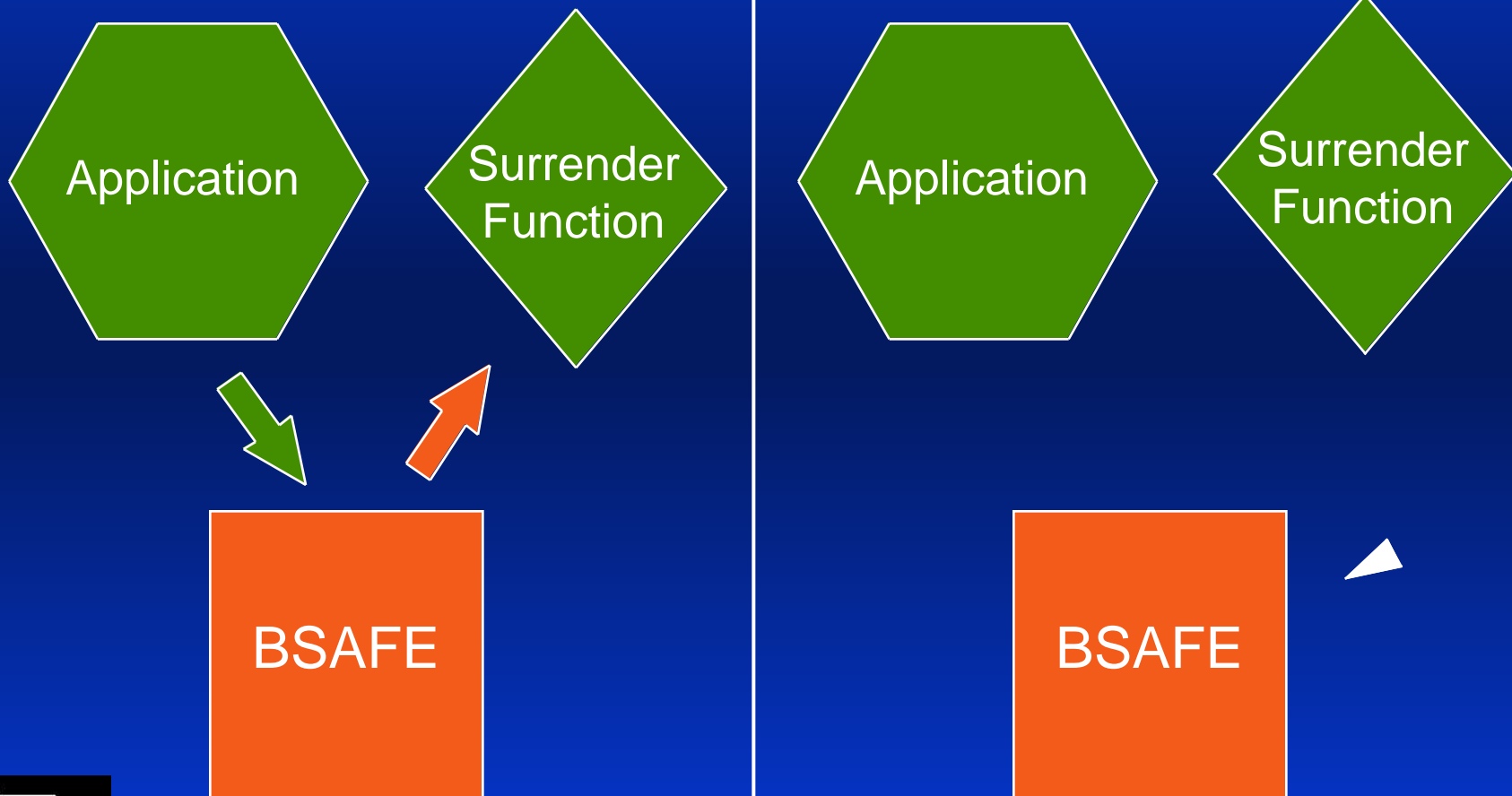
```
typedef unsigned char *POINTER;

typedef struct {
    int (*Surrender) ( );
    POINTER handle;
    POINTER reserved;
} A_SURRENDER_CTX;

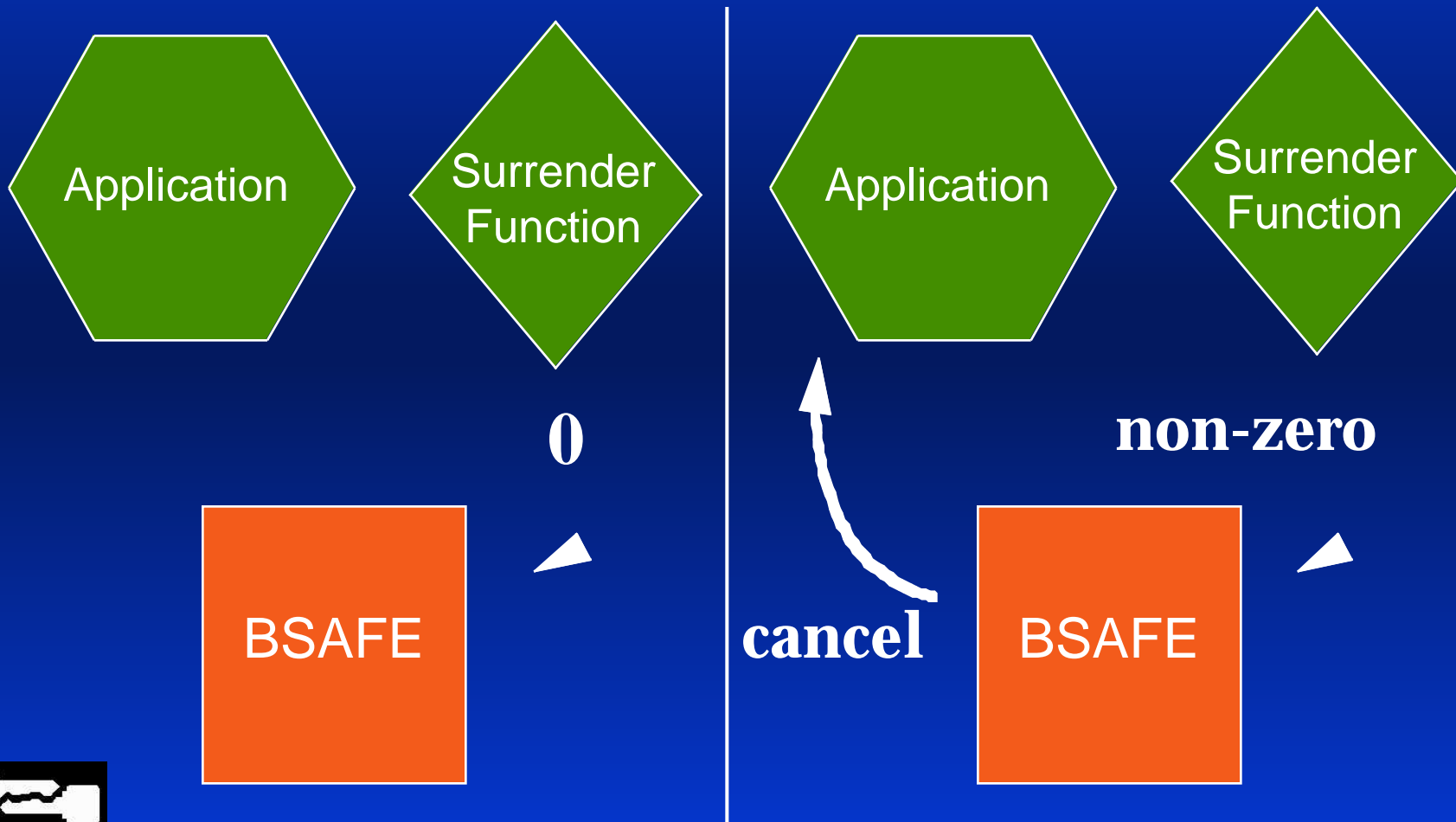
/* A return value of 0 means continue,
   a non-zero return value means cancel.
   */
int (*Surrender) (POINTER handle);
```



# Surrender



# Surrender



# Surrender Function

```
int MySurrenderFunction ( handle )
POINTER handle;
{
    UNUSED_ARG ( handle )

    /* _kbhit is a DOS/Windows call. */
    if ( _kbhit ( ) != 0 ) {
        _getch ( );
        return (1);
    }
    return (0);
}
```



# Surrender Context

```
A_SURRENDER_CTX mySurrenderContext;
```

```
mySurrenderContext.Surrender = MySurrenderFunction;
```

```
mySurrenderContext.handle = NULL_PTR;
```

```
mySurrenderContext.reserved = NULL_PTR;
```



# Surrender Context

```
if ((status = B_<function>
    (algorithmObject, <other arguments>,
    &mySurrenderContext)) != 0)
    break;
```



# Another Surrender Context

```
typedef struct {  
    int flag;  
    char message[80];  
} MY_HANDLE;
```

```
MY_HANDLE myHandle;
```



# Another Surrender Function

```
int MyNewSurrenderFunction ( handle )  
POINTER handle;  
{  
    if (((MY_HANDLE *)handle)->flag = 0) {  
        puts (((MY_HANDLE *)handle)->message);  
        ((MY_HANDLE *)handle)->flag = 1;  
    }  
    return (0);  
}
```



# Another Surrender Context

```
A_SURRENDER_CTX mySurrenderContext;
```

```
mySurrenderContext.Surrender = MyNewSurrenderFunction;  
mySurrenderContext.handle = (POINTER)&myHandle;  
mySurrenderContext.reserved = NULL_PTR;
```



# Another Surrender Context

```
myHandle.flag = 0;
strcpy (myHandle.message, "Begin RSA Encryption");

if ((status = B_<function>
    (algorithmObject, <other arguments>,
    &mySurrenderContext)) != 0)
    break;
```



# Security Considerations

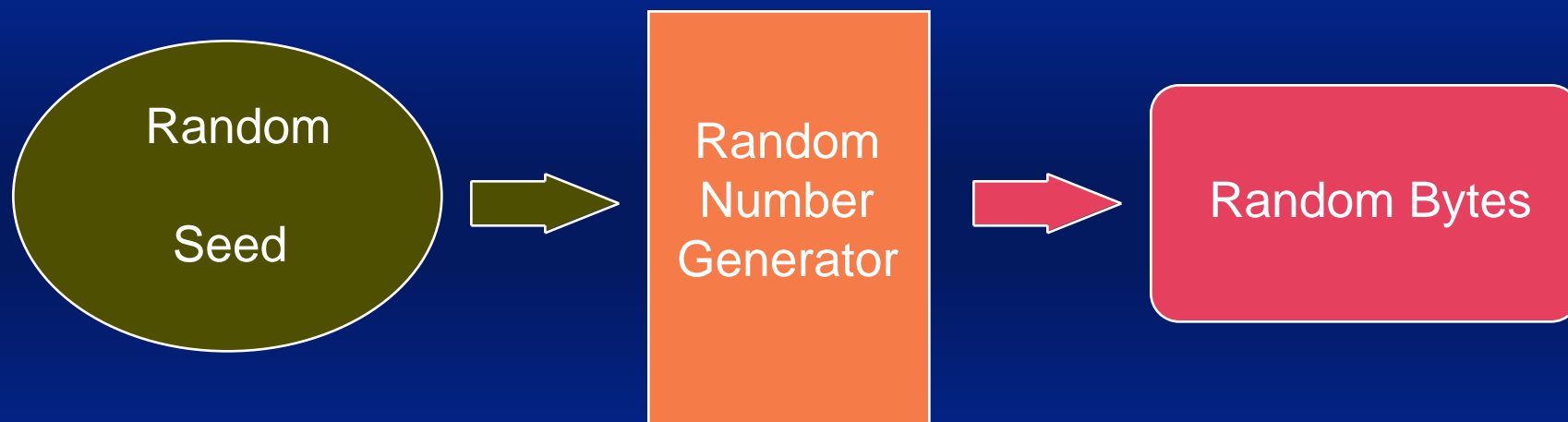
---

1. Random Number Seeding

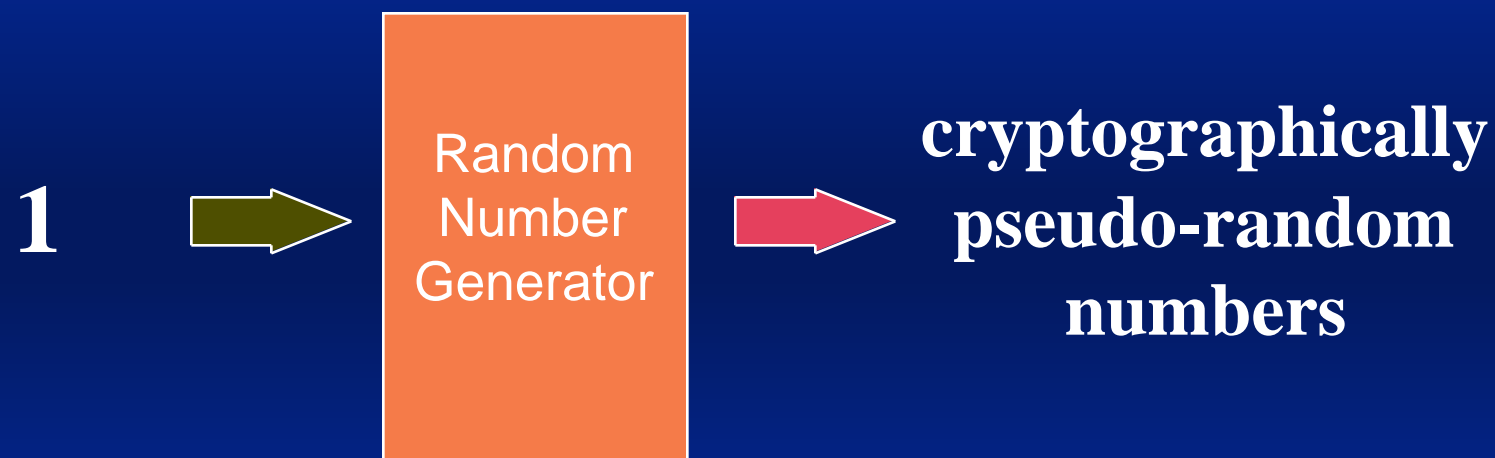
2. Stream Ciphers



# Random Number Seeding



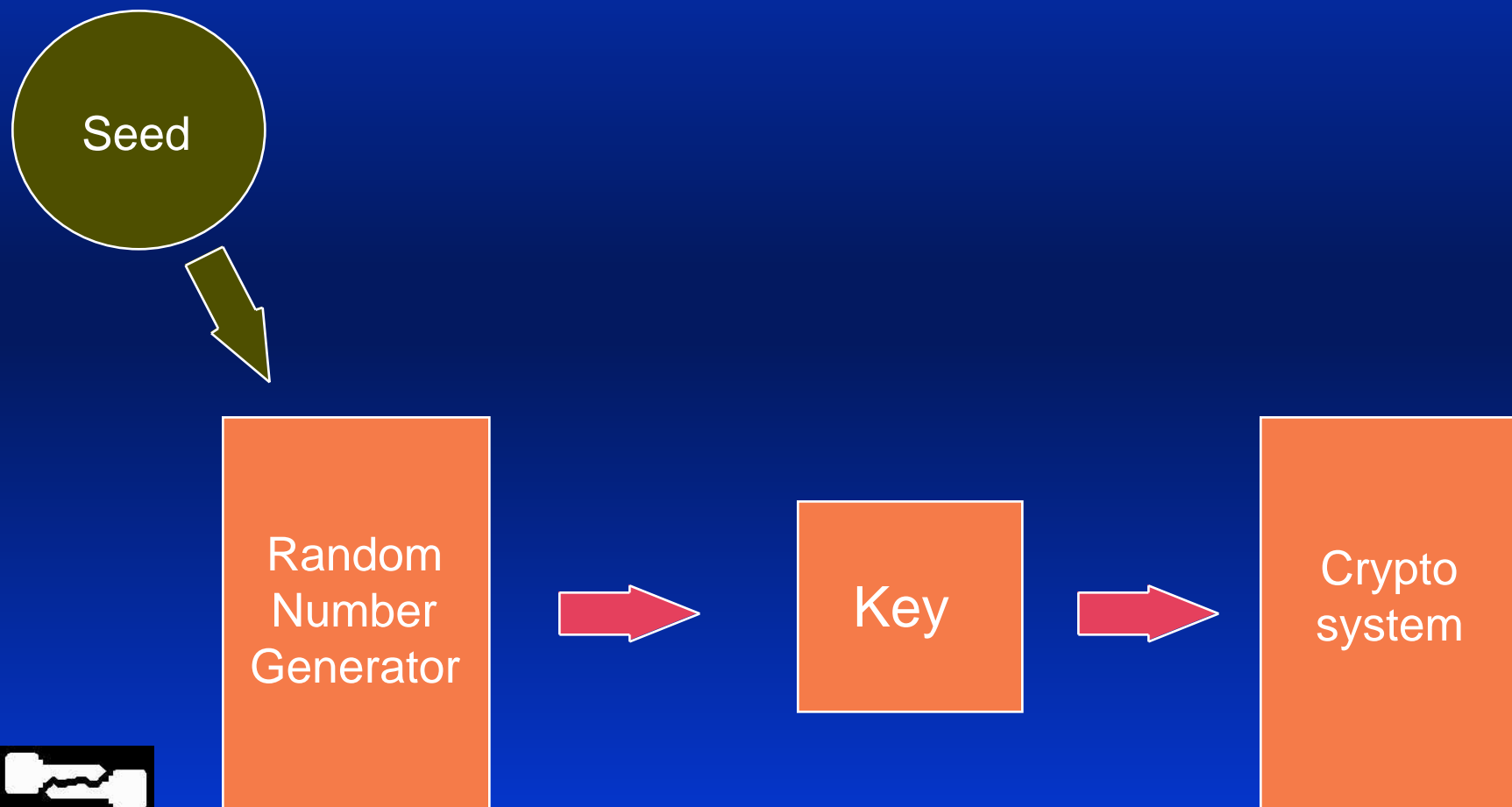
# Random Number Seeding



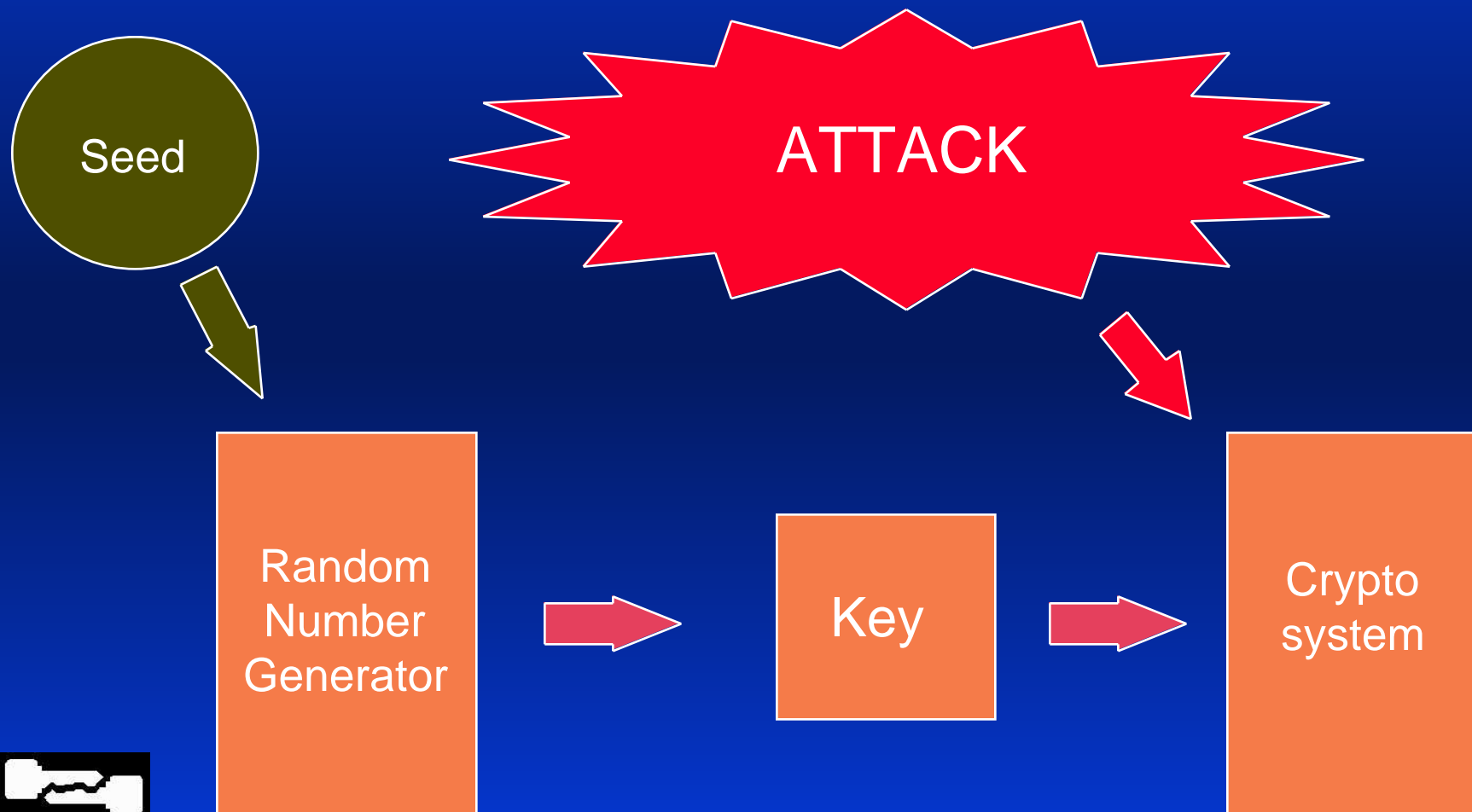
**A good random number generator does not depend on the seed for randomness, only unrepeatability.**



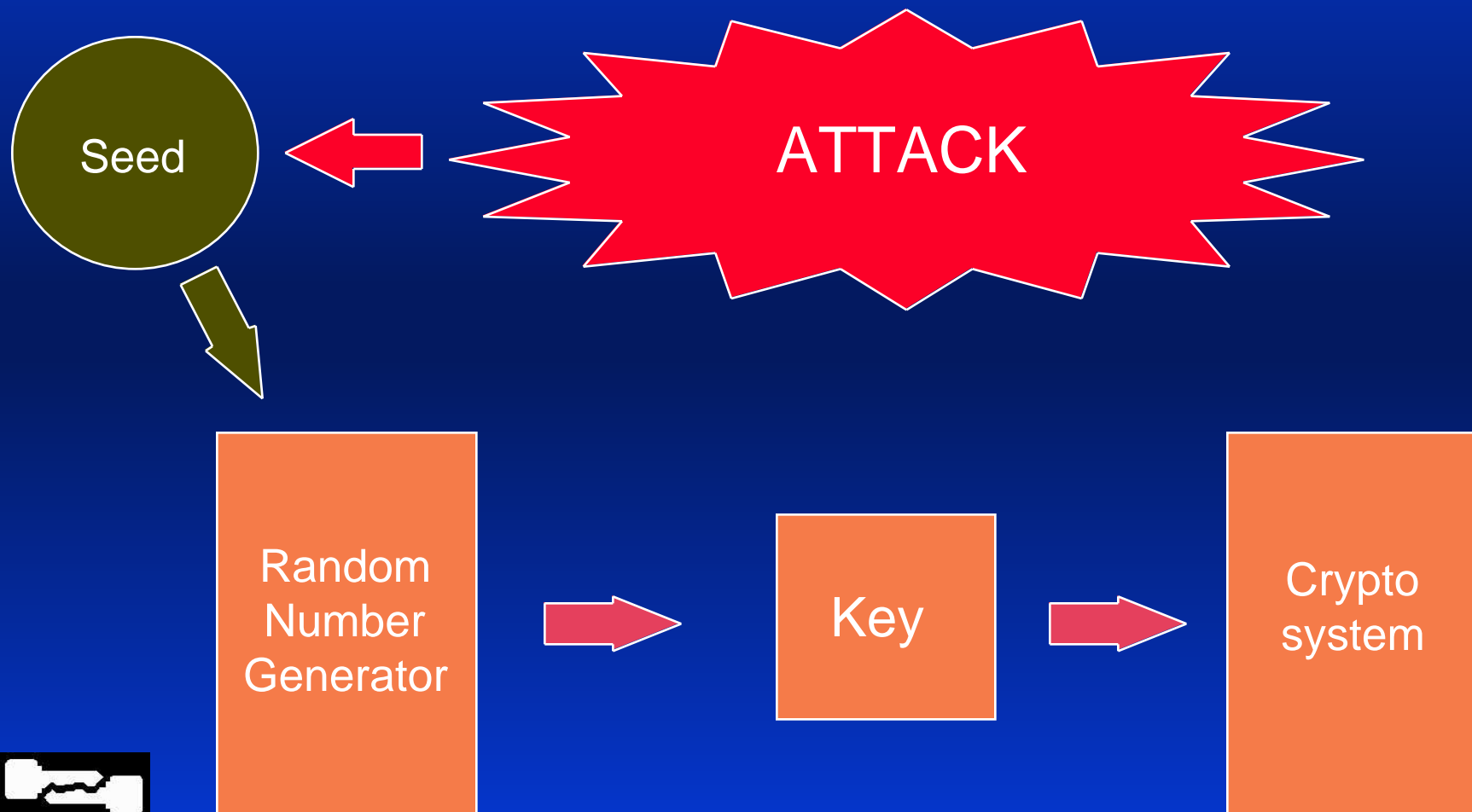
# Random Number Seeding



# Random Number Seeding



# Random Number Seeding



# Random Number Seeding

---

**The seed should be as strong as the key**

**Quality and Quantity**

**Quality for unrepeatability**

**Quantity to increase brute-force time**



# Random Number Seeding

---

**Quality = Entropy**

**Entropy: from order to chaos**

**Entropy: amount of information**

**Memory Statistics, Process Statistics,  
Mouse Movement, Keystroke Timing**



# Random Number Seeding

---

**Quantity**

**Rule of Thumb**

**At least  
1 byte of random seed  
for each  
1 bit of key**



# Stream Ciphers

---

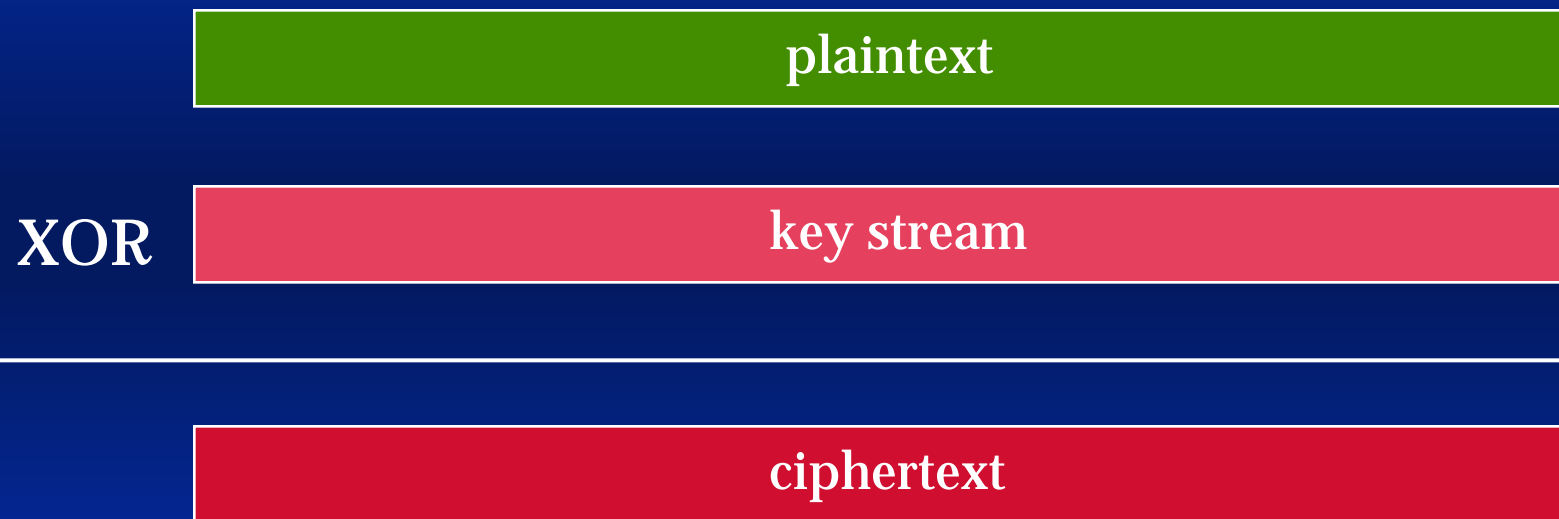
Generate a pseudo-random key stream

XOR the plaintext with the key stream  
to produce the ciphertext

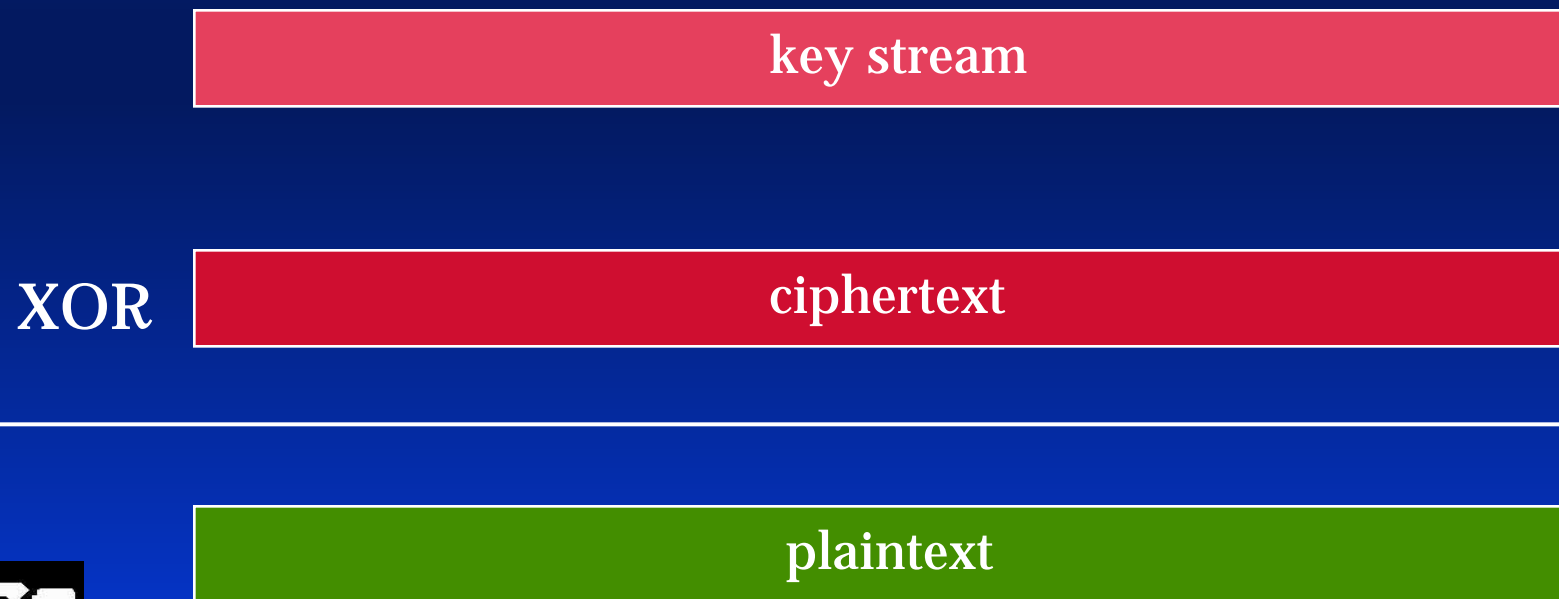
XOR the ciphertext with the same key  
stream to produce the plaintext



# Stream Ciphers



# Stream Ciphers



# Stream Ciphers

key stream

XOR

ciphertext

plaintext



# Stream Ciphers

Using the same key for two sessions is dangerous

plaintext 1

plaintext 2

key stream

key stream

ciphertext 1

ciphertext 2



# Stream Ciphers

Using the same key for two sessions is dangerous

plaintext 1

plaintext 2

key stream

key stream

ciphertext 1

ciphertext 2



# Stream Ciphers

	H	A	L
	0x48	0x41	0x4C
XOR	0x??	0x??	0x??
<hr/>			
	0x5C	0xF1	0xCA

	?	?	?
	0x??	0x??	0x??
XOR	0x??	0x??	0x??
<hr/>			
	0x5D	0xF2	0xCB



# Stream Ciphers

	H	A	L
	0x48	0x41	0x4C
XOR	0x14	0xB0	0x86
<hr/>			
	0x5C	0xF1	0xCA

	?	?	?
	0x??	0x??	0x??
XOR	0x??	0x??	0x??
<hr/>			
	0x5D	0xF2	0xCB



# Stream Ciphers

	H	A	L
	0x48	0x41	0x4C
XOR	0x14	0xB0	0x86
<hr/>			
	0x5C	0xF1	0xCA

	?	?	?
	0x??	0x??	0x??
XOR	0x14	0xB0	0x86
<hr/>			
	0x5D	0xF2	0xCB



# Stream Ciphers

    H    A    L  
    0x48 0x41 0x4C  
XOR 0x14 0xB0 0x86  
-----  
    0x5C 0xF1 0xCA

    I    B    M  
    0x49 0x42 0x4D  
XOR 0x14 0xB0 0x86  
-----  
    0x5D 0xF2 0xCB



# Stream Ciphers

Using the same key for two sessions is dangerous

plaintext 1

plaintext 2

key stream

key stream

ciphertext 1

ciphertext 2



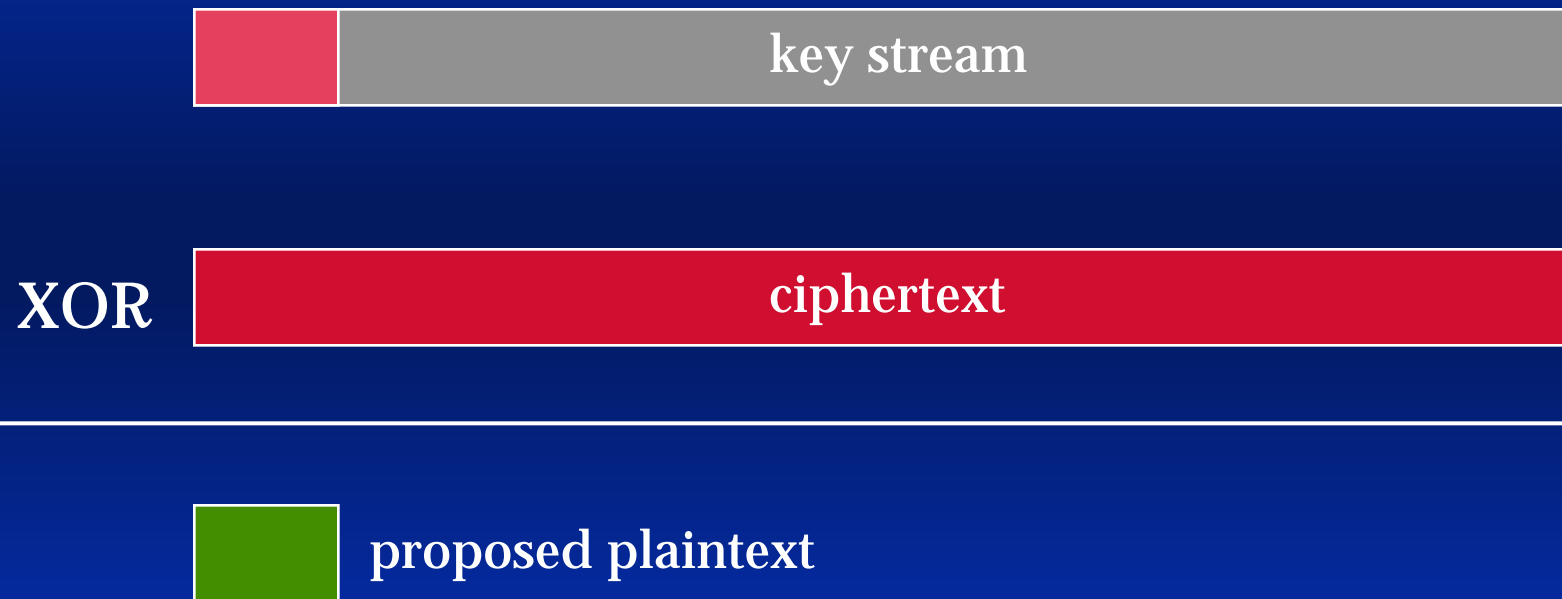
# Stream Ciphers, Exportable Keys

## Dictionary

Key	First n bytes of the Key Stream
0x00 0x00 ... 0x00	0x3C 0x29 ... 0x50
0x00 0x00 ... 0x01	0x64 0xF7 ... 0x11
.	.
.	.
.	.
0xff 0xff ... 0xff	0x25 0x5B ... 0xDE

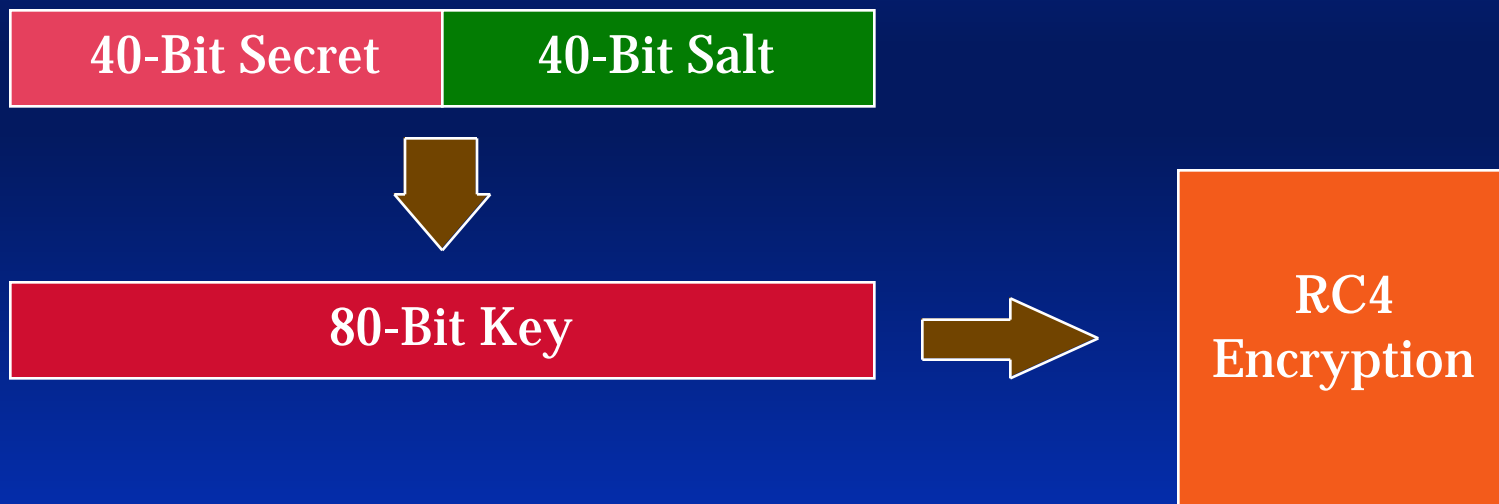


# Stream Ciphers, Exportable Keys

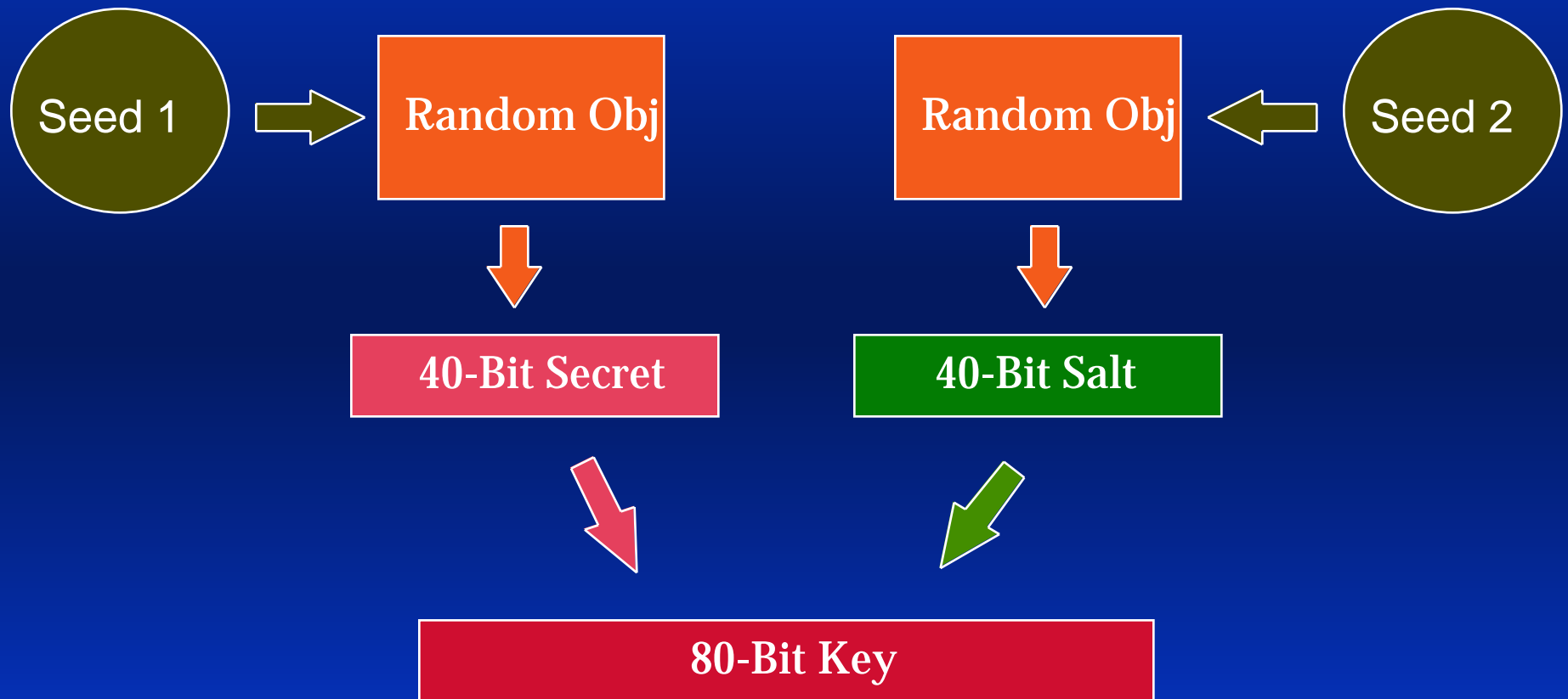


# Salting

RC4 Key = Secret Key + Salt



# Salting



# Salting

## Digital Envelope



# Using BSAFE

Steve Burnett  
RSA Data Security, Inc.  
burnetts@rsa.com

RSA Data Security Conference  
January 30, 1997

