

# BasicCard Crypto API

## C/C++ Programming Manual

Version: 1.01

Date: 31.03.00 18:15

Author: Michael Petig

email: [development@zeitcontrol.de](mailto:development@zeitcontrol.de)

Web sites:

<http://www.zeitcontrol.de>

<http://www.basiccard.com>

<http://www.basiccardusa.com>

## Contents

BasicCard Crypto API.....	1
C/C++ Programming Manual.....	1
1. Overview.....	3
2. Developer Guide.....	4
2.1 Setup Build Environment for C/C++ Compilers.....	4
2.2 Setup Runtime Environment.....	4
2.3 Redistribute Runtime Environment.....	4
2.4 Programming Intro.....	5
2.4.1 Header Files.....	5
2.4.2 Overview of Supported Function Groups.....	5
3. Reference.....	6
3.1 Functions.....	6
3.1.1 API Managment.....	6
3.1.2 DES and Triple-DES Functions.....	6
3.1.3 SHA.....	12
3.1.4 Random Functions.....	13
3.1.5 Elliptic Curve EC161.....	15

## 1. Overview

This document describes the BasicCard cryptographic support functions provided by our API. This API is designed to be used from C/C++ development environments like Microsoft Visual C++. For use with Microsoft Visual Basic an additional ActiveX control (OCX) is available which is based on this API. Refer to “BasicCard Crypto Control” “Visual Basic Programming Manual” for using this control.

The “BasicCard Crypto API” supports random generation, SHA, DES and EC161 (as implemented in BasicCard). It is designed to support BasicCard PC application developers by using cryptographic functions provided by BasicCard operating system and libraries. It is not designed to be used for applications unrelated to BasicCard. It may be used free without charge as long as it is used in conjunction with ZeitControl BasicCard. It is not allowed to use this API without using the BasicCard, too.

This document will describe how to use the supported cryptographic functions. We will not explain basic principles of using encryption nor will we describe BasicCard programming techniques in here. Please refer to additional literature like “Applied Cryptography” by “Bruce Schneier” for basic information about cryptography and “BasicCard, The Compact and Enhanced BasicCard” by “Tony Guilfoyle” (include in BasicCard Development Kit) for BasicCard programming and details of encryption used in BasicCard.

## 2. Developer Guide

### 2.1 Setup Build Environment for C/C++ Compilers

BasicCard Crypto API is included in “BasicCard API/OCX Developer Kit” so same setup rules apply.

Copy all header files (\*.h) from install C-API H directory (default: “C:\BasicCrd\API\H”) to your include directory, your project directory or extend the INCLUDE environment variable to include C-API H directory. Copy all library files (\*.lib) to your library directory, your project directory or extend the LIB environment variable to include C-API Lib directory (default: “C:\BasicCrd\API\Lib”). Include “zccrypt.h” in all source files which use BasicCard Crypto API functions. Link your executable and DLLs with “zccrypt.lib”. For details on how to apply these settings to your development environment refer to your C/C++ compiler reference.

To make libraries available for as many development systems as possible we have created libraries for several environments. This is necessary because most times libraries created for one compiler/linker will not work with a different set of compiler linker. So our install package includes libraries in following formats:

- Microsoft™ Visual C/C++ 5.0 and others (default: “C:\BasicCrd\API\Lib”)
- Microsoft™ new library format (default: “C:\BasicCrd\API\Lib\MS\_NEW”)
- Borland™ C-Builder (default: “C:\BasicCrd\API\Lib\Borland”)

If you still run into problems using import libraries “zccrypt.lib”, you may use the module definition file “zccrypt.def” to create your own import libraries.

Note: Since DLL entry points are exported by ordinal only and not by name you cannot use the DLL file itself to create an import library.

### 2.2 Setup Runtime Environment

After finishing the installation of the BasicCard API the required files are copied to your system directory. The BasicCard Crypto API runtime currently consists of one single DLL:

- ZCCRYPT.DLL

### 2.3 Redistribute Runtime Environment

We have decided to make it as easy as possible for you to include necessary files in your own setup programs. For this purpose we have decided to support new “Windows Installer” API. Take a look at [http://www.installsite.org/w2k\\_msiauth.htm](http://www.installsite.org/w2k_msiauth.htm) for products supporting this API. Following files (merge modules) may be used to be included in your own setup routines:

ZCCRYPT.MSM

## **2.4 Programming Intro**

### **2.4.1 Header Files**

You may find additional information by browsing through the header file:

ZCCRYPT.H constants, typedefs and function prototypes of BasicCard Crypto API

### **2.4.2 Overview of Supported Function Groups**

BasicCard Crypto API is split into several groups of functions. Each group starts with a different prefix.

#### **2.4.2.1 Global Functions**

Functions to manage this API itself. Currently only one function is included for this purpose:

ZCCryptGetVer

#### **2.4.2.2 DES and Triple-DES Functions**

This functions support DES and Triple-DES encryption in ECB (Electronic Code Book) and CBC (Cipher Block Chaining) mode. When encrypting large amount of data chaining (CBC) should be used. In CBC mode an IV (Initial Vector) is required. This should be chosen by random and stored with encrypted message.

Additionally DES and Triple-DES certificates are supported using CBC chaining as supported by BasicCard itself.

DES functions use prefix: ZCCryptDes...

#### **2.4.2.3 SHA**

This functions support SHA (Secure Hash).

SHA functions use prefix: ZCCryptSha...

#### **2.4.2.4 Random Generation**

Normal random values as generated by standard C function are not save to be used with encryption functions. So we have created an extended random generator. It is a difficult problem to create real secure and unpredictable random values using a standard computer. Our random generator is based on using a pool of random data which will be mixed using SHA with given seed and current random pool every time a new seed value is given to the random generator. Additionally a background thread used to collect as much “random” system data as possible to use this as seed to the random pool. Anyway we recommend to add additional random values as seed.

Random functions use prefix: ZCCryptRandom...

#### **2.4.2.5 Elliptic Curve EC161**

This functions support Elliptic Curve as implemented by BasicCard library included with BasicCard Development Kit V3.x. It is not compatible to old EC160 library as included with earlier versions of BasicCard development kit.

EC161 functions use prefix: ZCCryptEC161...

## 3. Reference

### 3.1 Functions

#### 3.1.1 API Managment

##### 3.1.1.1 ZCCryptGetVer

Get version of installed API DLL.

Note: Future versions may have extended functions which are not available in this or older versions of this API. You may use this function to check if a function you want to use is supported.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptGetVer(DWORD *pVer);
```

Parameter:

Number	Name	Direction	Description
1	*pVer	out	Version number of API. For example 0x100 if version 1.0.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.

#### 3.1.2 DES and Triple-DES Functions

##### 3.1.2.1 ZCCryptDesCreateContext

Create a context for use with DES encryption decryption operations.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesCreateContext(PZCCRYPT_CONTEXT_DES pContext, ZCCRYPT_ALGO algo, ZCCRYPT_MODE mode, const BYTE* pIV, const BYTE* pKey);
```

Parameter:

Number	Name	Direction	Description
1	* pContext,	out	Context to be used with future calls of ZCCryptDes.. functions.
2	algo	in	Either ZCCRYPT_ALGO_DES or ZCCRYPT_ALGO_3DES depending if DES or Triple-DES should be used.
3	mode	in	Chaining mode ZCCRYPT_MODE_ECB or ZCCRYPT_MODE_CBC. BasicCard DES function does no chaining (same as ECB mode) only a single block of 64 bits is encrypted. When encrypting more than 64 bit of data inside BasicCard it is recommended to make CBC or other chaining by implementing this chaining using Basic.
4	*pIV	in	IV (Initial Vector) of 8 bytes (64 bit) size to be used. Required for CBC mode. If NULL is passed a IV of 8 times 0 is used.
5	*pKey	in	Key to be used. Either 8 bytes (64 bit) for DES or 16 bytes (128 bit) for 3DES.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter e.g. pKey=NULL

### 3.1.2.2 ZCCryptDesSetIV

Change IV used in one context for example to use same context to encrypt and decrypt one message or to encrypt several messages. In this case you need to set the IV to a defined value before using context for a different block of encryption/decryption operations.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesSetIV(ZCCRYPT_CONTEXT_DES Context, const BYTE *pIV);
```

Parameter:

Number	Name	Direction	Description
1	Context	in	Context to as returned by ZCCryptDesCreateContext.
2	*pIV	in	New (8 byte) IV to set or NULL to set 8 times 0.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.2.3 ZCCryptDesGetIV

Get current active IV. When using chaining IV (for use with next block) changes after every block encrypted.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesGetIV(ZCCRYPT_CONTEXT_DES Context, BYTE *pIV);
```

Parameter:

Number	Name	Direction	Description
1	Context	in	Context to as returned by ZCCryptDesCreateContext.
2	*pIV	out	Current (8 byte) IV.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.2.4 ZCCryptDesDestroyContext

Destroy encryption context created by ZCCryptDesCreateContext. This call frees memory allocated for use by encryption and wipes secure data like assigned keys from memory (memset).

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesDestroyContext(PZCCRYPT_CONTEXT_DES pContext);
```

Parameter:

Number	Name	Direction	Description
1	*pContext	in	Context to as returned by ZCCryptDesCreateContext.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.2.5 ZCCryptDesEncrypt

Encrypt given data.

Note: You can encrypt a bigger block of data by calling ZCCyrptDesEncrypt several times starting with first part of data till reaching end of data. Than you must use the same unchanged context for every call.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesEncrypt(ZCCRYPT_CONTEXT_DES Context, void *pData,
DWORD cbLen);
```

Parameter:

Number	Name	Direction	Description
1	Context	in	Context to as returned by ZCCryptDesCreateContext.
2	pData	in	Pointer to data to encrypt
3	cbLen	in	Length of data in bytes. cbLen must be multiple of 8.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.2.6 ZCCryptDesDecrypt

Decrypt given data.

Note: You can decrypt a bigger block of data by calling ZCCyrptDesDecrypt several times starting with first part of data till reaching end of data. Than you must use the same unchanged context for every call.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesDecrypt(ZCCRYPT_CONTEXT_DES Context, void *pData,
DWORD cbLen);
```

Parameter:

Number	Name	Direction	Description
1	Context	in	Context to as returned by ZCCryptDesCreateContext.
2	pData	in	Pointer to data to decrypt
3	cbLen	in	Length of data in bytes. cbLen must be multiple of 8.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.

ZCCRYPT_ERROR_PARM	Invalid parameter (Context).
--------------------	------------------------------

### 3.1.2.7 ZCCryptDesEncryptOnce

Encrypt data with single function call. No need to create a context first when using this function.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesEncryptOnce(ZCCRYPT_ALGO algo, ZCCRYPT_MODE mode, const BYTE* pIV, const BYTE* pKey, void *pData, DWORD cbData);
```

Parameter:

Number	Name	Direction	Description
1	algo	in	Either ZCCRYPT_ALGO_DES or ZCCRYPT_ALGO_3DES depending if DES or Triple-DES should be used.
2	mode	in	Chaining mode ZCCRYPT_MODE_ECB or ZCCRYPT_MODE_CBC. BasicCard DES function does no chaining (same as ECB mode) only a single block of 64 bits is encrypted. When encrypting more than 64 bit of data inside BasicCard it is recommended to make CBC or other chaining by implementing this chaining using Basic.
3	*pIV	in	IV (Initial Vector) of 8 bytes (64 bit) size to be used. Required for CBC mode. If NULL is passed a IV of 8 times 0 is used.
4	*pKey	in	Key to be used. Either 8 bytes (64 bit) for DES or 16 bytes (128 bit) for 3DES.
5	pData	in	Pointer to data to encrypt
6	cbLen	in	Length of data in bytes. cbLen must be multiple of 8.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter e.g. pKey=NULL

### 3.1.2.8 ZCCryptDesDecryptOnce

Decrypt data with single function call. No need to create a context first when using this function.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesDecryptOnce(ZCCRYPT_ALGO algo, ZCCRYPT_MODE mode, const BYTE* pIV, const BYTE* pKey, void *pData, DWORD cbData);
```

Parameter:

Number	Name	Direction	Description
1	algo	in	Either ZCCRYPT_ALGO_DES or ZCCRYPT_ALGO_3DES depending if DES or Triple-DES should be used.
2	mode	in	Chaining mode ZCCRYPT_MODE_ECB or ZCCRYPT_MODE_CBC. BasicCard DES function does no chaining (same as ECB mode) only a single block of 64 bits is encrypted. When encrypting more than 64 bit of data inside BasicCard it is recommended to make CBC or other chaining by implementing this chaining using Basic.

3	*pIV	in	IV (Initial Vector) of 8 bytes (64 bit) size to be used. Required for CBC mode. If NULL is passed a IV of 8 times 0 is used.
4	*pKey	in	Key to be used. Either 8 bytes (64 bit) for DES or 16 bytes (128 bit) for 3DES.
5	pData	in	Pointer to data to decrypt
6	cbLen	in	Length of data in bytes. cbLen must be multiple of 8.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter e.g. pKey=NULL

### 3.1.2.9 ZCCryptDesStartCertificate

Start a DES or Triple-DES certificate (like BasicCard Certificate function).

Note: The context returned by this function must not be used with calls to standard DES (non certificate) functions. Also context created by ZCCryptDesCreateContext cannot be used with DES certificate functions.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesStartCertificate(PZCCRYPT_CONTEXT_DES_CERT pCert,
ZCCRYPT_ALGO algo, const BYTE* pKey);
```

Parameter:

Number	Name	Direction	Description
1	*pCert	out	Context to be used with following DES certificate calls.
2	algo	in	Either ZCCRYPT_ALGO_DES or ZCCRYPT_ALGO_3DES depending if DES or Triple-DES should be used.
3	*pKey	in	Key to be used. Either 8 bytes (64 bit) for DES or 16 bytes (128 bit) for 3DES.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.2.10 ZCCryptDesUpdateCertificate

Update certificate or in other words the certificate is calculated for data passed to this function.

Note: You may call this functions several times to certificate bigger amounts of data.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesUpdateCertificate(ZCCRYPT_CONTEXT_DES_CERT Cert,
const void *pData, DWORD cbData);
```

Parameter:

Number	Name	Direction	Description
1	Cert	in	Context as returned by ZCCryptDesStartCertificate.

2	pData	in	Pointer to data to certificate
3	cbData	in	Length of data in bytes. cbData can be any value e.g. 13.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.2.11 ZCCryptDesEndCertificate

Closes certificate context, free memory used and wipes secure data like keys. Additionally the resulting certificate is returned by this function.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesEndCertificate(PZCCRYPT_CONTEXT_DES_CERT pCert, void *pResult);
```

Parameter:

Number	Name	Direction	Description
1	*pCert	in	Context as returned by ZCCryptDesStartCertificate.
2	*pResult	out	Resulting certificate (8 bytes). pResult must point to valid memory buffer to store the resulting certificate.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.2.12 ZCCryptDesCertificateOnce

Calculate certificate by single function call. No need to call ZCCryptDesStartCertificate when using this function.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptDesCertificateOnce(ZCCRYPT_ALGO algo, const BYTE* pKey, const void *pData, DWORD cbData, void *pResult);
```

Parameter:

Number	Name	Direction	Description
1	algo	in	Either ZCCRYPT_ALGO_DES or ZCCRYPT_ALGO_3DES depending if DES or Triple-DES should be used.
2	*pKey	in	Key to be used. Either 8 bytes (64 bit) for DES or 16 bytes (128 bit) for 3DES.
3	pData	in	Pointer to data to certificate
4	cbData	in	Length of data in bytes. cbData can be any value e.g. 13.
5	*pResult	out	Resulting certificate (8 bytes). pResult must point to valid memory buffer to store the resulting certificate.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.3 SHA

#### 3.1.3.1 ZCCryptShaStart

Start SHA for data following by use of ZCCryptShaAppend.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptShaStart(PZCCRYPT_CONTEXT_SHA pSHAContext);
```

Parameter:

Number	Name	Direction	Description
1	*pSHAContext	out	Context to be used with further SHA calls.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

#### 3.1.3.2 ZCCryptShaAppend

Append data or in other words hash data passed to this call.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptShaAppend(ZCCRYPT_CONTEXT_SHA SHAContext, const void *pData, DWORD cbData);
```

Parameter:

Number	Name	Direction	Description
1	SHAContext	in	Context as returned by ZCCryptShaStart.
2	pData	in	Pointer to data to hash
3	cbData	in	Length of data in bytes. cbData can be any value e.g. 13.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

#### 3.1.3.3 ZCCryptShaEnd

Close SHA context and return resulting hash.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptShaEnd(PZCCRYPT_CONTEXT_SHA pSHAContext, void *pHash);
```

Parameter:

Number	Name	Direction	Description
--------	------	-----------	-------------

1	*pSHAContext	in	Context as returned by ZCCryptShaStart.
2	*pHash	out	Resulting hash (20 bytes). pHash must point to valid memory buffer to store the resulting hash.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.3.4 ZCCryptShaOnce

Do SHA with single function call. No need to call ZCCryptShaStart when using this function.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptShaOnce(const void *pData, DWORD cbData, void *pHash);
```

Parameter:

Number	Name	Direction	Description
1	pData	in	Pointer to data to hash
2	cbData	in	Length of data in bytes.
3	*pHash	out	Resulting hash (20 bytes). pHash must point to valid memory buffer to store the resulting hash.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

## 3.1.4 Random Functions

### 3.1.4.1 ZCCryptRandomInit

Initialize random support. If random is required, this function should be called as early as possible, so when query for random, random pool is well mixed (random).

Note: This function call will establish a background thread to seed random. You must call ZCCryptRandomTerm to close this thread before closing your application.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptRandomInit(void);
```

Parameter: none

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_MISC	Unknown error occurred.
ZCCRYPT_ERROR_RND_ALREADYINIT	Already initialized

### 3.1.4.2 ZCCryptRandomTerm

Terminate random support.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptRandomTerm(void);
```

Parameter: none:

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_MISC	Unknown error occurred.
ZCCRYPT_ERROR_RND_NOTINIT	Random not initialized

### 3.1.4.3 ZCCryptRandomSeed

Seed random pool. It is recommended to add own random, by use of this function.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptRandomSeed(const void *pSeed, DWORD cbSeed);
```

Parameter:

Number	Name	Direction	Description
1	pSeed	in	Pointer to data to use for seed
2	cbSeed	in	Length of data (pSeed) in bytes.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_MISC	Unknown error occurred.
ZCCRYPT_ERROR_RND_NOTINIT	Random not initialized
ZCCRYPT_ERROR_PARM	Invalid parameter.

### 3.1.4.4 ZCCryptRandom

Get some random bytes. ZCCryptRandomInit must have been called before this function can be used.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptRandom(void *pRandom, DWORD cbRandom);
```

Parameter:

Number	Name	Direction	Description
1	*pRandom	out	Random data. pRandom must point to valid memory to store data.
2	cbRandom	in	Length of requested random in bytes. (Max 40 bytes)

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_MISC	Unknown error occurred.
ZCCRYPT_ERROR_RND_NOTINIT	Random not initialized
ZCCRYPT_ERROR_PARM	Invalid parameter.

### 3.1.5 Elliptic Curve EC161

#### 3.1.5.1 ZCCryptEC161CreateContext

Create ECC context from predefined BasicCard curve.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptEC161CreateContext(PZCCRYPT_CONTEXT_EC161 pEC161, int iCurve);
```

Parameter:

Number	Name	Direction	Description
1	*pEC161	out	Context to be used with further calls to ZCCryptEC161.. function calls.
2	iCurve	in	Number (1 to 3) of predefined curve to use.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter.
ZCCRYPT_ERROR_MEM	Not enough memory

#### 3.1.5.2 ZCCryptEC161CreateContextFromParams

Create ECC context from curve parameters. This function could be used when curve parameters (stored in EC161Params) are loaded from BasicCard. For this purpose you can use ZCBCIUSER as parameter type in conjunction with pointer to ZCCRYPT\_EC161DOMAINPARAMS variable when retrieving parameters from BasicCard through BasicCard API.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptEC161CreateContextFromParams (PZCCRYPT_CONTEXT_EC161 pEC161, const ZCCRYPT_EC161DOMAINPARAMS *pParams);
```

Parameter:

Number	Name	Direction	Description
1	*pEC161	out	Context to be used with further calls to ZCCryptEC161.. function calls.
2	*pParams	in	Curve parameters.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter.
ZCCRYPT_ERROR_MEM	Not enough memory
ZCCRYPT_ERROR_EC161_BADPARAMS	Bad ecc parameters (*pParams)

#### 3.1.5.3 ZCCryptEC161DestroyContext

Destroy context created by ZCCryptEC161CreateContext or ZCCryptEC161CreateContextFromParams.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptEC161DestroyContext(PZCCRYPT_CONTEXT_EC161 pEC161);
```

Parameter:

Number	Name	Direction	Description
1	*pEC161	in	Context as returned by ZCCryptEC161CreateContext or ZCCryptEC161CreateContextFromParams.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).

### 3.1.5.4 ZCCryptEC161GetPublicKey

Get public key from known private key.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptEC161GetPublicKey(ZCCRYPT_CONTEXT_EC161 EC161,
PZCCRYPT_EC161PRIVATEKEY pPrivateKey, PZCCRYPT_EC161PUBLICKEY pPublicKey);
```

Parameter:

Number	Name	Direction	Description
1	EC161	in	Context as returned by ZCCryptEC161CreateContext or ZCCryptEC161CreateContextFromParams.
2	*pPrivateKey	in	21 byte private key.
		out	21 byte corrected private key
3	*pPublicKey	out	22 byte public key. pPublicKey must point to valid memory to store key.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).
ZCCRYPT_ERROR_EC161_BADPARAMS	Bad ecc parameters.

### 3.1.5.5 ZCCryptEC161GenerateKeyPair

Generate a new key pair (private key and public key).

Note: To use this function ZCCryptRandomInit must have been called before. This function uses ZCCryptRandom to generate a random private key. Before calling ZCCryptRandom, \*pSeed is used to seed random pool.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptEC161GenerateKeyPair(ZCCRYPT_CONTEXT_EC161 EC161,
PZCCRYPT_EC161PRIVATEKEY pPrivateKey, PZCCRYPT_EC161PUBLICKEY pPublicKey, void *pSeed,
DWORD cbSeed);
```

Parameter:

Number	Name	Direction	Description
1	EC161	in	Context as returned by ZCCryptEC161CreateContext or ZCCryptEC161CreateContextFromParams.
2	*pPrivateKey	out	21 byte private key. pPrivateKey must point to valid memory to store key.

3	*pPublicKey	out	22 byte public key. pPublicKey must point to valid memory to store key.
4	*pSeed	in	Random seed to use for generating key. Note: You may pass in NULL instead, but it is strongly recommended to add some random data (which should not be generated by ZCCryptRandom) by use of this parameter.
5	cbSeed	in	Size of seed (*pSeed).

Return values:

Value	Description
ZCCRYPT_NOERROR	No error.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).
ZCCRYPT_ERROR_RND_NOTINIT	Random generator not initialized.

### 3.1.5.6 ZCCryptEC161Verify

Verify a ECC signature of 20 byte message or hash.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptEC161Verify(ZCCRYPT_CONTEXT_EC161 EC161, const
ZCCRYPT_EC161SIGNATURE *pSignature, const ZCCRYPT_EC161PUBLICKEY *pPublicKey, const void
*pMessage);
```

Parameter:

Number	Name	Direction	Description
1	EC161	in	Context as returned by ZCCryptEC161CreateContext or ZCCryptEC161CreateContextFromParams.
2	*pSignature	in	42 byte signature.
3	*pPublicKey	in	Signer's 22 byte public key.
4	*pMessage	in	20 byte message to be verified.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error, signature valid.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).
ZCCRYPT_ERROR_EC161_BADPARAMS	Invalid public key
ZCCRYPT_ERROR_EC161_BADSIGNATURE	Invalid signature or in other words verify failed.

### 3.1.5.7 ZCCryptEC161HashAndVerify

Verify ECC signature of message by first hash message using SHA and then verify signature by use of resulting 20 byte hash.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptEC161HashAndVerify(ZCCRYPT_CONTEXT_EC161 EC161,
const ZCCRYPT_EC161SIGNATURE *pSignature, const ZCCRYPT_EC161PUBLICKEY *pPublicKey, const
void *pMessage, DWORD cbMessage);
```

Parameter:

Number	Name	Direction	Description
--------	------	-----------	-------------

1	EC161	in	Context as returned by ZCCryptEC161CreateContext or ZCCryptEC161CreateContextFromParams.
2	*pSignature	in	42 byte signature.
3	*pPublicKey	in	Signer's 22 byte public key.
4	*pMessage	in	Message to be verified.
5	cbMessage	in	Length of message to be verified.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error, signature valid.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).
ZCCRYPT_ERROR_EC161_BADPARAMS	Invalid public key
ZCCRYPT_ERROR_EC161_BADSIGNATURE	Invalid signature or in other words verify failed.

### 3.1.5.8 ZCCryptEC161SharedSecret

Calculate the shared secret.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptEC161SharedSecret(ZCCRYPT_CONTEXT_EC161 EC161, const
ZCCRYPT_EC161PRIVATEKEY *pPrivateKey, const ZCCRYPT_EC161PUBLICKEY *pPublicKey,
PZCCRYPT_EC161SHAREDSECRET pSecret);
```

Parameter:

Number	Name	Direction	Description
1	EC161	in	Context as returned by ZCCryptEC161CreateContext or ZCCryptEC161CreateContextFromParams.
2	*pPrivateKey	in	Own 21 byte private key.
3	*pPublicKey	in	Other party's 22-byte public key.
4	*pSecret	out	20 byte shared secret.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error, signature valid.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).
ZCCRYPT_ERROR_EC161_BADPARAMS	Invalid key.

### 3.1.5.9 ZCCryptEC161SessionKey

Calculate session key.

Note: You can use a session key to establish a encrypted connection between two parties. Session key generated here must therefor be used with symmetric encryption like DES.

Syntax:

```
ZCCRYPTRET ZCCRYPTLINK ZCCryptEC161SessionKey(ZCCRYPT_CONTEXT_EC161 EC161, const
ZCCRYPT_EC161PRIVATEKEY *pPrivateKey, const ZCCRYPT_EC161PUBLICKEY *pPublicKey, const
void *pKdp, DWORD cbKdp, PZCCRYPT_EC161SESSIONKEY pKey);
```

Parameter:

Number	Name	Direction	Description
1	EC161	in	Context as returned by ZCCryptEC161CreateContext or ZCCryptEC161CreateContextFromParams.
2	*pPrivateKey	in	Own 21 byte private key.
3	*pPublicKey	in	Other party's 22-byte public key.
4	*pKdp	in	Key derivation parameter.
5	cbKdp	in	Length of key derivation parameter.
6	*pKey	out	20 byte session key.

Return values:

Value	Description
ZCCRYPT_NOERROR	No error, signature valid.
ZCCRYPT_ERROR_PARM	Invalid parameter (Context).
ZCCRYPT_ERROR_EC161_BADPARAMS	Invalid key.

## 3.2 Constants and Enumeration

### 3.2.1 Error Codes

Value (Hex)	Symbolic name	Description
0	ZCCRYPT_NOERROR	No error, function call succeeded.
0x03000000	ZCCRYPT_ERROR_PARM	Invalid parameter passed to function call.
0x03000001	ZCCRYPT_ERROR_MISC	Unknown, unexpected error.
0x03000002	ZCCRYPT_ERROR_MEM	Not enough memory.
0x03000003	ZCCRYPT_ERROR_NOTALLOWED	Call not allowed now, maybe some parameter not set in context.
0x03000004	ZCCRYPT_ERROR_MISSINGPARG	One or more parameter missing.
0x03001000	ZCCRYPT_ERROR_EC161_BADPARAMS	Bad ECC parameters.
0x03001001	ZCCRYPT_ERROR_EC161_BADSIGNATURE	Verify signature failed, signature is invalid.
0x03002000	ZCCRYPT_ERROR_RND_NOTINIT	Random not initialized.
0x03002001	ZCCRYPT_ERROR_RND_ALREADYINIT	Random already initialized.

### 3.2.2 Enumeration ZCCRYPT\_ALGO

Enumeration type to choose encryption algorithm.

C typedef:

```
typedef enum _ZCCRYPT_ALGO {
    ZCCRYPT_ALGO_DES=0x21,
    ZCCRYPT_ALGO_3DES=0x22,
    ZCCRYPT_ALGO_DUMMY=0x7FFFFFFF /* force enum to use 32 bit values */
} ZCCRYPT_ALGO;
```

### 3.2.3 Enumeration ZCCRYPT\_MODE

Enumeration to select encryption chaining mode.

C typedef:

```
typedef enum _ZCCRYPT_MODE {
    ZCCRYPT_MODE_ECB=1,
    ZCCRYPT_MODE_CBC=2,
    ZCCRYPT_MODE_DUMMY=0x7FFFFFFF /* force enum to use 32 bit values */
} ZCCRYPT_MODE;
```

### 3.3 Type Definitions

#### 3.3.1 ZCCRYPTRET

API return value. See 3.2.1 Error Codes.

C typedef:

```
typedef DWORD ZCCRYPTRET;
typedef ZCCRYPTRET *PZCCRYPTRET;
```

#### 3.3.2 ZCCRYPT\_CONTEXT\_DES

DES encryption context

C typedef:

```
typedef void *ZCCRYPT_CONTEXT_DES;
typedef ZCCRYPT_CONTEXT_DES *PZCCRYPT_CONTEXT_DES;
```

#### 3.3.3 ZCCRYPT\_CONTEXT\_DES\_CERT

DES certificate context

C typedef:

```
typedef void *ZCCRYPT_CONTEXT_DES_CERT;
typedef ZCCRYPT_CONTEXT_DES_CERT *PZCCRYPT_CONTEXT_DES_CERT;
```

#### 3.3.4 ZCCRYPT\_CONTEXT\_SHA

SHA context.

C typedef:

```
typedef void *ZCCRYPT_CONTEXT_SHA;
typedef ZCCRYPT_CONTEXT_SHA *PZCCRYPT_CONTEXT_SHA;
```

#### 3.3.5 ZCCRYPT\_EC161DOMAINPARAMS

ECC domain parameters as used in BasicCard.

C typedef:

```
typedef struct _ZCCRYPT_EC161DOMAINPARAMS
{
    BYTE a ;
    BYTE b[21] ;
    BYTE r[21] ;
    BYTE k ;
    BYTE Gx[21] ;
    BYTE Gy ;
} ZCCRYPT_EC161DOMAINPARAMS;
typedef ZCCRYPT_EC161DOMAINPARAMS *PZCCRYPT_EC161DOMAINPARAMS;
```

#### 3.3.6 ZCCRYPT\_EC161PRIVATEKEY

ECC private key.

C typedef:

```
typedef BYTE ZCCRYPT_EC161PRIVATEKEY[21];
typedef BYTE *PZCCRYPT_EC161PRIVATEKEY;
```

### 3.3.7 ZCCRYPT\_EC161PUBLICKEY

ECC public key.

C typedef:

```
typedef BYTE ZCCRYPT_EC161PUBLICKEY[22];  
typedef BYTE *PZCCRYPT_EC161PUBLICKEY;
```

### 3.3.8 ZCCRYPT\_EC161SIGNATURE

ECC signature.

C typedef:

```
typedef BYTE ZCCRYPT_EC161SIGNATURE[42];  
typedef BYTE *PZCCRYPT_EC161SIGNATURE;
```

### 3.3.9 ZCCRYPT\_EC161SHAREDSECRET

ECC shared secret.

C typedef:

```
typedef BYTE ZCCRYPT_EC161SHAREDSECRET[21];  
typedef BYTE *PZCCRYPT_EC161SHAREDSECRET;
```

### 3.3.10 ZCCRYPT\_EC161SESSIONKEY

ECC session key.

C typedef:

```
typedef BYTE ZCCRYPT_EC161SESSIONKEY[20];  
typedef BYTE *PZCCRYPT_EC161SESSIONKEY;
```

### 3.3.11 ZCCRYPT\_CONTEXT\_EC161

ECC context.

C typedef:

```
typedef void *ZCCRYPT_CONTEXT_EC161;  
typedef ZCCRYPT_CONTEXT_EC161 *PZCCRYPT_CONTEXT_EC161;
```