Advanced Encryption Standard (AES)

Simon Heron, Internet Security Analyst, Network Box

Simon Heron

Ever since humans started talking, we have been trying to work out a way of communicating information to one person, while keeping it secret from everyone else. Certainly, when you start to look at encryption methods, you are introduced to the 'Caesar' Cypher. which shows how old encryption is and how simple old ciphers now are for us to break.

We have seen plenty of evidence of this since the introduction of the Data Encryption Standard (DES) in the 70s, its various incarnations since then, and most recently its upgrade to Triple DES in 1999 (which happened as attacks on the original cipher became increasingly viable as computing resources grew). However, even Triple DES has had its day and has, since 2002, been superseded by the Advanced Encryption Standard (AES) that we should all be using. But keeping encryption technology ahead of the game is a continual process, and security professionals are already investigating the successor to AES.

"Even Triple DES has had its day and has, since 2002, been superseded by the Advanced Encryption Standard (AES) that we should all be using"

This is, inevitably, a moving target; it has to be to keep up with computing power and increasingly sophisticated mathematical analysis. But AES is still a long way from being obsolete. There are still many solutions that are using its predecessor 3DES. So in this paper, I will look at how AES works, without going into the maths.

Explanation

Before going in to the detail, here are definitions of some of the terms I will be using:

Plaintext: The original message and the one that is to be encrypted.

Ciphertext: The encrypted message which is the output of the algorithm.

Block Size: The number of bits that the cipher will operate on. This is 128 bits in AES. Each message is split up into this number of bits and the operations described below carried out.

Even Triple DES has had its day and has, since 2002, been superseded by the Advanced Encryption Standard (AES) that we should all be using"

AES has a fixed block size of 128 bits. So all data that is to be encrypted will be broken up into 128 bit blocks. Padding is added if the data is not a multiple of 128bits. For manipulation purposes, 128 bits = 16 bytes (8 * 16 = 128) which is then treated as a 4 x 4 byte matrix.

This is termed the 'state'. This is important as the subsequent algorithm will manipulate this 'state' to create the ciphertext.

Encryption Key Size: This is the length in bits of the key used to encrypt or decrypt the message. In this standard the cipher key can only be 128, 192 or 256 bits long.

There are some differences in the way the algorithm works for the different encryption key lengths, so this paper will primarily focus on the cipher key of 128 bits.

Some basic principles

There are three important issues for an encryption algorithm:

Only the encryption key is secret
The algorithm can be made public but that does not help an attacker decipher a message as the encryption key is required to implement the algorithm. This allows us to define a standard which everybody can follow.

Confusion
This involves obscuring the relationship between the plaintext and the ciphertext. So in its simplest form, it is substituting one letter for a completely different one. The trick is to make that relationship impenetrable.

A	n	p	n
n	c	t	k
space	r	i	e
e	y	o	y

Figure 2: The Encryption Key.

01000001	01110011	01100101	01100111
00100000	01110100	01110011	01100101
01110100	00100000	01110011	00000000
01100101	01101101	01100001	00000000

=

A	s	e	g
space	t	s	e
t	space	s	padding
e	m	a	padding

Figure 1: The State.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 1: S-box data is generally provided as a look-up table.

0x8d,	0x01,	0x02,	0x04,	0x08,	0x10,	0x20,	0x40,
0x80,	0x1b,	0x36,	0x6c,	0xd8,	0xab,	0x4d,	0x9a,
0x2f,	0x5e,	0xbc,	0x63,	0xc6,	0x97,	0x35,	0x6a,
0xd4,	0xb3,	0x7d,	0xfa,	0xef,	0xc5,	0x91,	0x39,
0x72,	0xe4,	0xd3,	0xbd,	0x61,	0xc2,	0x9f,	0x25,
0x4a,	0x94,	0x33,	0x66,	0xcc,	0x83,	0x1d,	0x3a,
0x74,	0xe8,	0xcb,	0x8d,	0x01,	0x02,	0x04,	0x08,
0x10,	0x20,	0x40,	0x80,	0x1b,	0x36,	0x6c,	0xd8,
0xab,	0x4d,	0x9a,	0x2f,	0x5e,	0xbc,	0x63,	0xc6,
0x97,	0x35,	0x6a,	0xd4,	0xb3,	0x7d,	0xfa,	0xef,
0xc5,	0x91,	0x39,	0x72,	0xe4,	0xd3,	0xbd,	0x61,
0xc2,	0x9f,	0x25,	0x4a,	0x94,	0x33,	0x66,	0xcc,
0x83,	0x1d,	0x3a,	0x74,	0xe8,	0xcb,	0x8d,	0x01,
0x02,	0x04,	0x08,	0x10,	0x20,	0x40,	0x80,	0x1b,
0x36,	0x6c,	0xd8,	0xab,	0x4d,	0x9a,	0x2f,	0x5e,
0xbc,	0x63,	0xc6,	0x97,	0x35,	0x6a,	0xd4,	0xb3,
0x7d,	0xfa,	0xef,	0xc5,	0x91,	0x39,	0x72,	0xe4,
0xd3,	0xbd,	0x61,	0xc2,	0x9f,	0x25,	0x4a,	0x94,
0x33,	0x66,	0xcc,	0x83,	0x1d,	0x3a,	0x74,	0xe8,
0xcb,	0x8d,	0x01,	0x02,	0x04,	0x08,	0x10,	0x20,
0x40,	0x80,	0x1b,	0x36,	0x6c,	0xd8,	0xab,	0x4d,
0x9a,	0x2f,	0x5e,	0xbc,	0x63,	0xc6,	0x97,	0x35,
0x6a,	0xd4,	0xb3,	0x7d,	0xfa,	0xef,	0xc5,	0x91,
0x39,	0x72,	0xe4,	0xd3,	0xbd,	0x61,	0xc2,	0x9f,
0x25,	0x4a,	0x94,	0x33,	0x66,	0xcc,	0x83,	0x1d,
0x3a,	0x74,	0xe8,	0xcb,	0x8d,	0x01,	0x02,	0x04,
0x08,	0x10,	0x20,	0x40,	0x80,	0x1b,	0x36,	0x6c,
0xd8,	0xab,	0x4d,	0x9a,	0x2f,	0x5e,	0xbc,	0x63,
0xc6,	0x97,	0x35,	0x6a,	0xd4,	0xb3,	0x7d,	0xfa,
0xef,	0xc5,	0x91,	0x39,	0x72,	0xe4,	0xd3,	0xbd,
0x61,	0xc2,	0x9f,	0x25,	0x4a,	0x94,	0x33,	0x66,
0xcc,	0x83,	0x1d,	0x3a,	0x74,	0xe8,	0xcb	

Table 2: The look-up table that results from the RCON function.

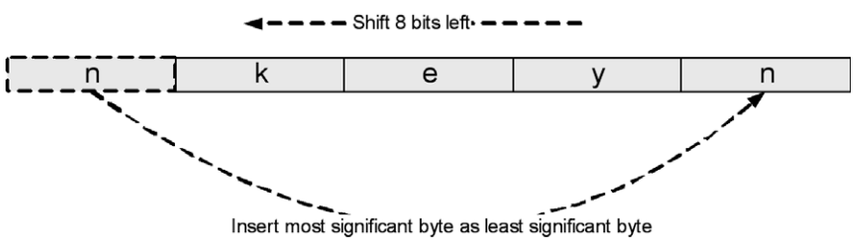

Figure 3: Shift Function.

Diffusion

This is the mixing and reordering of the message. So operations here would be shifting data or transposing columns, for instance.

Rijndael's Substitution Box (S-Box)

This is the most important function that performs substitution which obscures the relationship between the plaintext and the ciphertext. It introduces the second of the three requirements above: confusion.

The S-Box is created by using a form of modulus mathematics which is called Rijndael's Galois field and within this field, arithmetic has special properties which ensure values do not exceed 2^8 which keeps everything in a byte, which is great for computers. The choice of this field and other steps that are taken to derive the S-Box are carefully chosen to resist cryptanalysis and are definitely beyond the scope of this article. In practice, S-box is generally used in the form of a lookup table:

The thing to hang onto here is that to process a number through the S-Box, each number is divided into its most and least significant nibble (4 bits). The least significant nibble identifies the column to use in the above table and the most significant nibble defines the row.

An example

For this, we have to move to hexadecimal which is frequently indicated by prefixing a number with '0x' so that 16 in base 10 (decimal) becomes 0x10 in base 16 (hexadecimal).

So to convert 0x53, divide into 0x50 and 0x03 and at the intersection of row and column we find: 0xed. Similarly, 0xe5 becomes 0xd9.

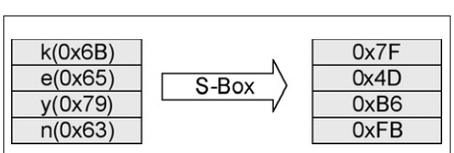

Figure 4: S-Box step.

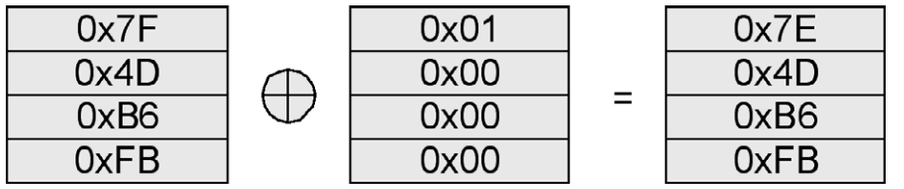

Figure 5: 1st Round so Rcon step, rcon(1) = 0x01.

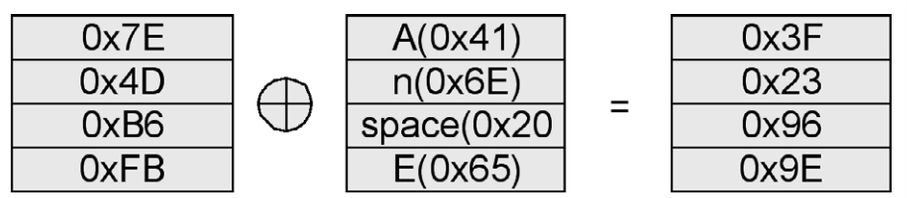

Figure 6: Creating the first column of the new key.

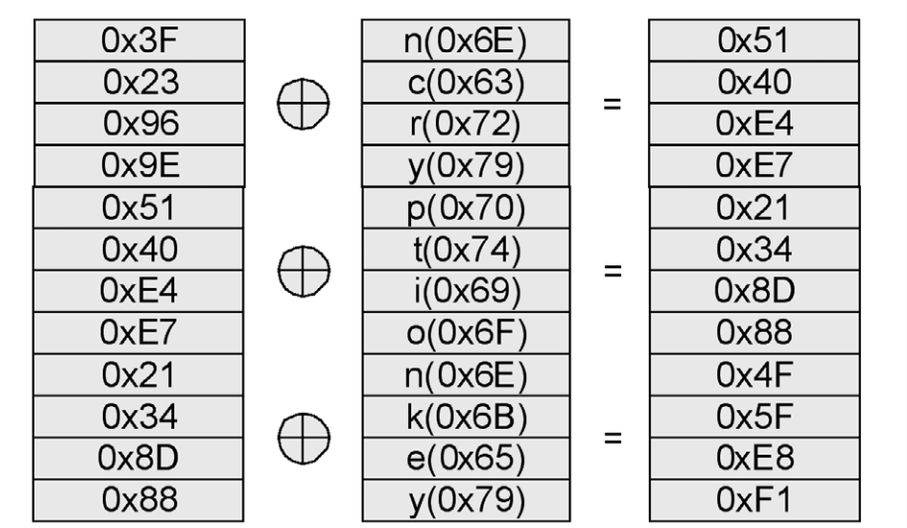

Figure 7: Creating round key for round 1.

Rcon

This function is used to confuse the derivations of the encryption key that will be used in the standard. Very simplistically, this function is putting 2 to the power of 254 to 509 but in the Rijndael's Galois field which uses its form of mathematics to keep values within a byte. The result is another look up table.

This is used to create a number of encryption keys (or round keys). Each key size (128, 192 and 256 bits) has a different number of 'rounds'. Each round requires a different key to provide the required diffusion and hence a different number of round keys need to be derived from the encryption key.

For AES128, the first 'n' bytes are the original key itself. Then the last column of the key is taken and round shifted (as shown in figure 3, above).

The result is then pushed through the s-box to provide a completely new column:

The most significant byte of the result is then XOR-ed with a value from the Rcon table above. For the first round, the index into the Rcon table will be 1, and then is incremented for each new round.

To create the first column of the next round key, the result of this XOR is XOR-ed with the first column of the previous round key. In the case of the first round, this is:

The next three columns of the new 'key' are created by taking each new column created and XOR-ing it with the four byte block 16 bytes before the new expansion key.

This is repeated until there are enough keys for each round. For AES128, there are 10 rounds and so 10 extra keys are required.

Enough groundwork. Now lets start the encryption.

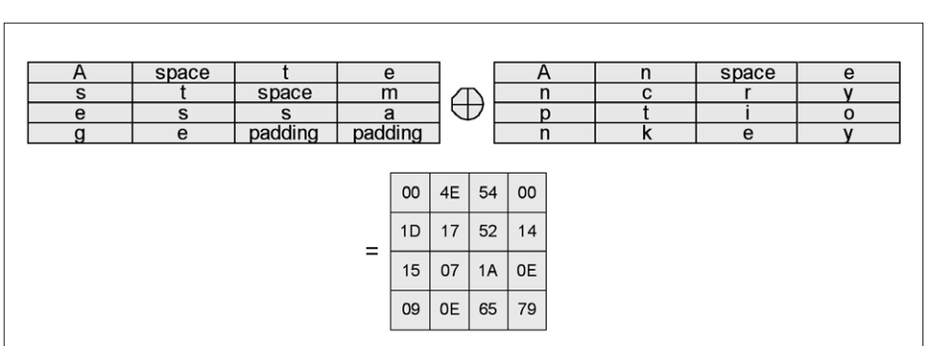

Figure 8: The initial round.

The Advanced Encryption Standard

There are three main stages in the standard:

The initial round (this is really just initialisation).
The intermediate rounds
The final round

- **The initial round**
  This is straightforward: take the plain-text and XOR it with the encryption key.
- **The intermediate rounds**
  This stage has four steps:

`SubBytes` – use the s-box to replace each byte of the state with a new value.

`ShiftRows` – Rotate the state in a prescribed fashion.

`MixColumns` – Take 4 bytes and mix them so that all four bytes have an effect on the value of each of the resulting four bytes.

`AddRoundKey` – each byte of the state is combined with the round key created by the key schedule above.

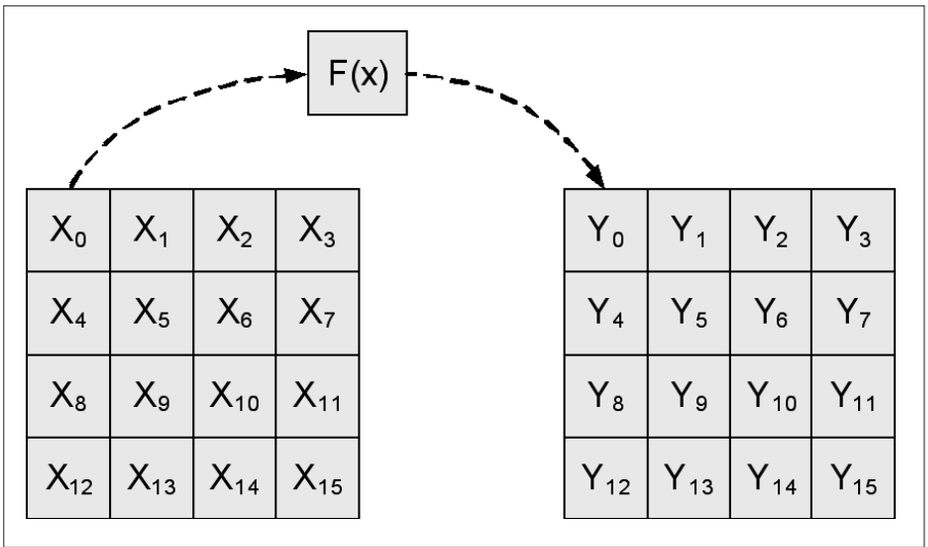

Figure 9: Use the S-Box to create 'confusion'.

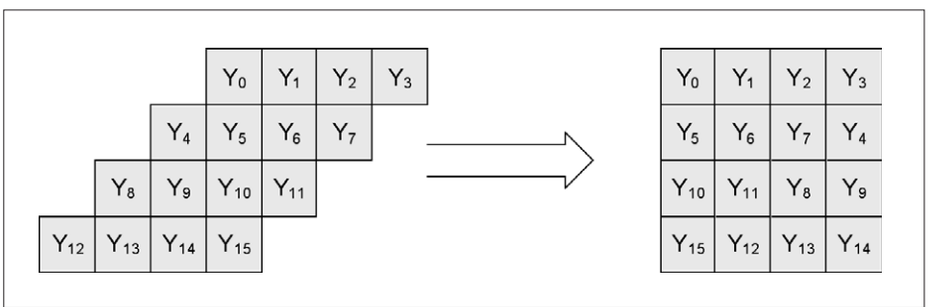

Figure 10: Shift Rows step.

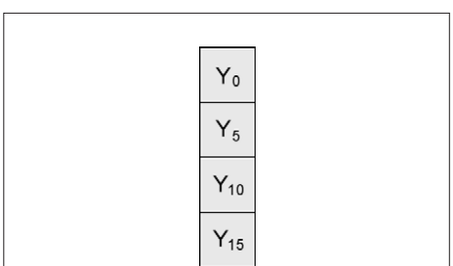

Figure 11: The resulting four input bytes from the row shifting step.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} y0 \\ y5 \\ y10 \\ y15 \end{bmatrix} = \begin{bmatrix} x0 \\ x4 \\ x8 \\ x12 \end{bmatrix}$$

Figure 12: Matrix produced by polynomial multiplation of four input bytes (as in listing 1).

Let us look at each of these steps in a bit more detail.

Step 1: SubBytes
Take each byte and using Rijndael's S-Box algorithm map that byte to the new value.

Step 2: ShiftRows
The next two steps now introduce 'diffusion' into the standard. For AES128, each row is shifted left by the value of the row where the top row is row zero. So the top row stays as it is, the first row is shift 1 to the left, the second row is shifted two to the left and the third row three to the left.

Step 3: MixColumns
In this step the four bytes of each column of the state are used as input. So, for example, from figure 10, the first four bytes to be used are:
This step takes these four bytes and multiplies them in Rijndael's Galois field by a fixed polynomial:

$$c(x) = 3x^3 + x^2 + x + 2 \text{ modulo } x^4 + 1$$

This results in the following matrix which may make things clearer:

Since this is done in Rijndael's Galois field, the addition is just exclusive OR (^) but the multiplication is followed by dividing by a reducing polynomial $x^8 + x^4 + x^3 + x + 1$ to keep it within the finite field. In the following, the symbol "•" is used to denote this type of multiplication:

$$x_0 = 2•Y_0 \text{ ^ } 3•Y_5 \text{ ^ } Y_{10} \text{ ^ } Y_{15}$$
$$x_4 = Y_0 \text{ ^ } 2•Y_5 \text{ ^ } 3•Y_{10} \text{ ^ } Y_{15}$$
$$x_8 = Y_0 \text{ ^ } Y_5 \text{ ^ } 2•Y_{10} \text{ ^ } 3•Y_{15}$$
$$x_{12} = 3•Y_0 \text{ ^ } Y_5 \text{ ^ } Y_{10} \text{ ^ } 2•Y_{15}$$

The result, r_x, satisfies the requirement to ensure that all four rows impact on the result. As this step uses XOR and is carefully designed, it can be reduced to the lines of C outlined in listing 1:

Step 4: AddRoundKey
This is the same as the initial round above, but using one of the keys derived from the Rijndael's key schedule. So, the result from Step 3, MixColumns, is XOR-ed with the next in the sequence of keys generated by the Rijndael's Key Schedule so each round is XOR-ed with a different key derived from the original.

Repeat the steps
Repeat steps 1 to 4 until the number of 'rounds' defined for the keysize by the standard have been carried out for this stage:

AES128 – repeat 9 times
AES192 – repeat 11 times
AES256 – repeat 13 times

This leaves one round for the final step.

- **The final round**
  There are more attacks being devised against AES and as computing capacity improves, developments of these attacks may become practicable"

This round carries out three steps:

```
SubBytes
ShiftRows
AddRoundKey
```

The reason why the final round does not have a 'mixcolumns' step is because that step is used to feed into the next round. Since this is the final step and there is no next round, the final round excludes that step.

```
void gmix_column(unsigned char *r) {
        unsigned char a[4];
        unsigned char b[4];
        unsigned char c;
        unsigned char h;
        /* The array 'a' is simply a copy of the input array 'r'
        * The array 'b' is each element of the array 'a' multiplied by
        * in Rijndael's Galois field
        * a[n] ^ b[n] is element n multiplied by 3 in Rijndael's Galois field */
        for(c=0;c<4;c++) {
                a[c] = r[c];
                h = r[c] & 0x80; /* hi bit */
                b[c] = r[c] << 1;
                if(h == 0x80)
                        b[c] ^= 0x1b; /* Rijndael's Galois field */
        }
        r[0] = b[0] ^ a[3] ^ a[2] ^ b[1] ^ a[1]; /* 2 * a0 + a3 + a2 + 3 * a1 */
        r[1] = b[1] ^ a[0] ^ a[3] ^ b[2] ^ a[2]; /* 2 * a1 + a0 + a3 + 3 * a2 */
        r[2] = b[2] ^ a[1] ^ a[0] ^ b[3] ^ a[3]; /* 2 * a2 + a1 + a0 + 3 * a3 */
        r[3] = b[3] ^ a[2] ^ a[1] ^ b[0] ^ a[0]; /* 2 * a3 + a2 + a1 + 3 * a0 */
}
```

Listing 1: C code designed to mix columns in a Rijndael Galois field. (Code provided by: http://en.wikipedia.org/wiki/Rijndael_mix_columns).

That is it, the state is encrypted.

Conclusion

There are more attacks being devised against AES and as computing capacity improves, developments of these attacks may become practicable. However, AES can always up the number of rounds or move to creating dynamic S-boxes to improve the resistance of the standard. The main attack is usually against implementations of the standard (where someone has programmed the standard and made a mistake that can be exploited). It is recommended that properly validated and supported code is used – of which there are a number of options – to protect against this sort of attack.

About the author

Simon Heron is an internet security analyst at Network Box (UK) Ltd, a managed security company, where he is responsible for developing the overall business and technology strategy and growth.

Heron has more than 16 years experience in the IT industry, including eight years experience in internet security. During this time he has developed and designed technologies ranging from firewalls, anti-virus, LANs and WANs.

Prior to Network Box, Heron co-founded and was Technical Director of Cresco Technologies Ltd, a network design and simulation solution company with customers in the USA, Europe and China. Before that he worked for Microsystems Engineering Ltd, as a Project Manager, where he implemented network security for the company.

Heron began his career as a digital hardware and software engineer, developing pioneering speech recognition technology before moving on to work for the British Antarctic Survey (B.A.S.) as science project leader. While at the B.A.S. he spent two Antarctic winters at the research station Halley in the Antarctic, developing and enhancing graphical technologies in the harshest of conditions.

Heron has an MSc in Microprocessor Technology and Applications, and a BSc in Naval Architecture and Shipbuilding and is a Certified Information Systems Security Professional (CISSP).

A complement to the GridOne authentication method

Seung S Yang and Hongsik Choi

Seung S Yang

Hongsik Choi

Authentication is the process of confirming that something claimed is true, and an authentication method is the way of proving the claim. As more and more people use networked computer systems in their everyday lives – for example, in online banking, email, and online shopping – the importance of authentication is increasing.

Three types of authentication methods are frequently used: identification (ID) device-based, biometric-based and password-based. A password-based authentication method is easier to implement and deploy, and costs less to manage than the other two. The method relies on a possession of knowledge