Exploring Cryptography Using Sage

Minh Van Nguyen
nguyenminh2@gmail.com

Victoria University
11 December 2009
Melbourne, Australia

Contents

Cryptography & computer algebra systems

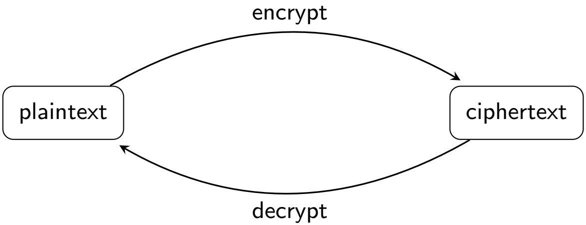

Figure 1: The encryption/decryption cycle.

- What's a computer algebra system (CAS)?
- FriCAS (Axiom & OpenAxiom), Magma, Maple, Mathematica, Matlab, Maxima, Sage

CAS in cryptography education

Closed source CASs (mid-1990s to present)

- Baliga and Boztas [1]: Maple for engineering & information security; Magma for advanced computational algebra
- Cosgrave [2]: Maple; applications of number theory to cryptography
- Eisenberg [3]: Mathematica; application of linear algebra to cryptography
- Klima et al. [4]: Maple and Matlab, plus custom code. Applications of algebra to cryptography.
- May [6]: Maple worksheets
- Trappe and Washington [10]: Maple, Mathematica and Matlab, plus custom code

Open source CASs (mid-2000s to present)

- Kohel [5]: Sage; initial developer of cryptography module
- McAndrew [7]: Axiom and Maxima for computer exercises

Cryptography specific functionalities

functionality	FriCAS	Maple	Math.	Matlab	Maxima	Sage
affine		KSS, TW	TW	KSS, TW		✓
Caesar	AM	KSS, MM, TW	TW	KSS, TW	AM	✓
Hill	AM	KSS, MM, TW	TW	KSS, TW	AM	✓
shift		KSS, TW	TW	KSS, TW		✓✓
substitution						✓
transposition						✓
Vigenère	AM	KSS	TW	KSS, TW	AM	✓

Table 1: Classical cryptosystems.

Legends

- ✓ = supported by the CAS
- ✓ = our implementation
- AM = code by McAndrew [7]
- KSS = code by Klima et al. [4]
- MM = code by May [6]
- TW = code by Trappe and Washington [10]

Cryptography specific functionalities

functionality	FriCAS	Maple	Math.	Matlab	Maxima	Sage
Euler phi	✓	✓	✓	TW	✓	✓
extended GCD	✓	✓	✓	✓	AM	✓
GCD	✓	✓	✓	✓	✓	✓
factorization	✓	✓	✓	✓	✓	✓
inverse mod. arith.	✓	✓	✓		✓	✓
modular arith.	✓	✓	✓	✓	✓	✓
modular exp.	✓	✓	✓	TW	✓	✓
n-th prime	AM	✓	✓	✓	AM	✓
next prime	✓	✓	✓		✓	✓
previous prime	✓	✓			✓	✓
primality test	✓	✓	✓	TW	✓	✓

Table 2: Number theoretic functionalities.

Cryptography specific functionalities

functionality	FriCAS	Maple	Math.	Matlab	Maxima	Sage
DSS						✓
ElGamal	AM				AM	
MD5						✓
Rabin	AM				AM	
RSA	AM	KSS, MM, TW	TW	KSS, TW	AM	
SHA						✓

Table 3: Hashing and digital signatures.

Cryptography specific functionalities

functionality	FriCAS	Maple	Math.	Matlab	Maxima	Sage
Merkle-Hellman	AM				AM	
subset sum problem	AM				AM	
super-increasing sequences	AM				AM	✓

Table 4: Knapsack cryptosystems.

functionality	FriCAS	Maple	Math.	Matlab	Maxima	Sage
AES		KSS, MM				✓✓
DES	AM	MM			AM	✓✓
elliptic curves		KSS	TW	KSS, TW		✓
finite fields	✓	✓	✓		✓	✓
S-box						✓

Table 5: Support for AES, DES, finite fields and elliptic curves.

The shift cryptosystem

Encryption & decryption

- Encryption function $\mathcal{E} : \mathbf{Z}/n\mathbf{Z} \times \mathbf{Z}/n\mathbf{Z} \longrightarrow \mathbf{Z}/n\mathbf{Z}$ given by $\mathcal{E}(k, p) = p + k \pmod{n}$
- Decryption function $\mathcal{D} : \mathbf{Z}/n\mathbf{Z} \times \mathbf{Z}/n\mathbf{Z} \longrightarrow \mathbf{Z}/n\mathbf{Z}$ given by $\mathcal{D}(k, c) = c - k \pmod{n}$

Cryptanalysis

- Exhaustive key search; key space $\mathbf{Z}/n\mathbf{Z}$ has n keys
- Statistical key ranking by squared-differences rank

$$R_{RSS}(\mathbf{P}_k) = \sum_{e \in \mathcal{A}} \left(O_{\mathbf{P}_k}(e) - E_{\mathcal{A}}(e) \right)^2$$

or by chi-square rank

$$R_{\chi^2}(\mathbf{P}_k) = \sum_{e \in \mathcal{A}} \frac{\left(O_{\mathbf{P}_k}(e) - E_{\mathcal{A}}(e) \right)^2}{E_{\mathcal{A}}(e)}$$

Shift cryptosystem: Sage examples

`sage.crypto.classical.ShiftCryptosystem`

Encryption & decryption

```
sage: S = ShiftCryptosystem(AlphabeticStrings()); S
Shift cryptosystem on Free alphabetic string monoid on A-Z
sage: plaintext = S.encoding("Shift cryptosystem generalizes Caesar cipher.")
sage: plaintext
SHIFTCRYPTOSYSTEMGENERALIZESCAESARCIPHER
sage: key = 7
sage: ciphertext = S.enciphering(key, plaintext); ciphertext
ZOPMAJYFWAVZFZALTNLULYHSPGLZJHLZHYJPWOLY
sage: S.deciphering(key, ciphertext)
SHIFTCRYPTOSYSTEMGENERALIZESCAESARCIPHER
sage: S.deciphering(key, ciphertext) == plaintext
True
```

Exhaustive key search

```
sage: candidates = S.brute_force(ciphertext)
sage: sorted(candidates.items())
[(0, ZOPMAJYFWAVZFZALTNLULYHSPGLZJHLZHYJPWOLY),
 (1, YNOLZIXEVZUYEYZKSMKTKXGROFKYIGKYGXIOVNKX),
 (2, XMNKYHWDUYTXDXYJRLJSJWFQNEJXHFJXFWHNUMJW),
...  # and so on
```

Shift cryptosystem: Sage examples

Key ranking by squared-differences method

```
sage: ranked_cand = S.brute_force(ciphertext, ranking="squared_differences")
sage: ranked_cand[:5]  # top five candidate keys
[(7, SHIFTCRYPTOSYSTEMGENERALIZESCAESARCIPHER),
 (11, ODEBPYNULPKOUOPAICAJANWHEVAOYWAOWNYELDAN),
 (18, HWXUIRGNEIDHNHITBVTCTGPAXOTHRPTHPGRXEWTG),
 (22, DSTQENCJAEZDJDEPXRPYPCLWTKPDNLPDLCNTASPC),
 (20, FUVSGPELCGBFLFGRZTRARENYVMRFPNRFNEPVCURE)]
```

Key ranking by chi-square method

```
sage: ranked_cand = S.brute_force(ciphertext, ranking="chisquare")
sage: ranked_cand[:5]  # top five candidate keys
[(7, SHIFTCRYPTOSYSTEMGENERALIZESCAESARCIPHER),
 (11, ODEBPYNULPKOUOPAICAJANWHEVAOYWAOWNYELDAN),
 (20, FUVSGPELCGBFLFGRZTRARENYVMRFPNRFNEPVCURE),
 (4, VKLIWFUBSWRVBVWHPJHQHUDOLCHVFDHVDUFLSKHU),
 (13, MBCZNWLSJNIMSMNYGAYHYLUFCTYMWUYMULWCJBYL)]
```

The affine cryptosystem

Encryption & decryption

- Encryption function $\mathcal{E} : (\mathbf{Z}/n\mathbf{Z})^* \times \mathbf{Z}/n\mathbf{Z} \times \mathbf{Z}/n\mathbf{Z} \longrightarrow \mathbf{Z}/n\mathbf{Z}$ given by $\mathcal{E}(k, p) = ap + b \pmod{n}$, where $k = (a, b) \in (\mathbf{Z}/n\mathbf{Z})^* \times \mathbf{Z}/n\mathbf{Z}$
- Decryption function $\mathcal{D} : (\mathbf{Z}/n\mathbf{Z})^* \times \mathbf{Z}/n\mathbf{Z} \times \mathbf{Z}/n\mathbf{Z} \longrightarrow \mathbf{Z}/n\mathbf{Z}$ given by $\mathcal{D}(k, c) = a^{-1}(c - b) \pmod{n}$, where $k = (a^{-1}, -a^{-1}b) \in (\mathbf{Z}/n\mathbf{Z})^* \times \mathbf{Z}/n\mathbf{Z}$

Cryptanalysis

- Exhaustive key search; key space $(\mathbf{Z}/n\mathbf{Z})^* \times \mathbf{Z}/n\mathbf{Z}$ has $\varphi(n) \times n$ keys
- Statistical key ranking by squared-differences rank

$$R_{RSS}(\mathbf{P}_k) = \sum_{e \in \mathcal{A}} \left(O_{\mathbf{P}_k}(e) - E_{\mathcal{A}}(e) \right)^2$$

or by chi-square rank

$$R_{\chi^2}(\mathbf{P}_k) = \sum_{e \in \mathcal{A}} \frac{\left(O_{\mathbf{P}_k}(e) - E_{\mathcal{A}}(e) \right)^2}{E_{\mathcal{A}}(e)}$$

Affine cryptosystem: Sage examples

`sage.crypto.classical.AffineCryptosystem`

Encryption & decryption

```
sage: A = AffineCryptosystem(AlphabeticStrings()); A
Affine cryptosystem on Free alphabetic string monoid on A-Z
sage: plaintext = A.encoding("Affine cryptosystem generalizes shift cipher.")
sage: plaintext
AFFINECRYPTOSYSTEMGENERALIZESSHIFTCIPHER
sage: a, b = (9, 13)
sage: ciphertext = A.enciphering(a, b, plaintext); ciphertext
NGGHAXFKVSCJTVTCXRPXAXKNIHEXTTYHGCFHSYXK
sage: A.deciphering(a, b, ciphertext)
AFFINECRYPTOSYSTEMGENERALIZESSHIFTCIPHER
sage: A.deciphering(a, b, ciphertext) == plaintext
True
```

Exhaustive key search

```
sage: candidates = A.brute_force(ciphertext)
sage: sorted(candidates.items())
[((1, 0), NGGHAXFKVSCJTVTCXRPXAXKNIHEXTTYHGCFHSYXK),
 ((1, 1), MFFGZWEJURBISUSBWQOWZWJMHGDWSSXGFBEGRXWJ),
 ((1, 2), LEEFYVDITQAHRTRAVPNVYVILGFCVRRWFEADFQWVI),
...  # and so on
```

Affine cryptosystem: Sage examples

Key ranking by squared-differences method

```
sage: ranked_cand = A.brute_force(ciphertext, ranking="squared_differences")
sage: ranked_cand[:5]  # top five candidate keys
[((9, 13), AFFINECRYPTOSYSTEMGENERALIZESSHIFTCIPHER),
 ((1, 19), UNNOHEMRCZJQACAJEYWEHERUPOLEAAFONJMOZFER),
 ((5, 6), RAAVETFGDSULNDNUTXHTETGRQVKTNNOVAUFVSOTG),
 ((3, 7), CRRAPOIBWVHSEWEHOMUOPOBCJAZOEEXARHIAVXOB),
 ((23, 5), GRRITUAHMNBQEMEBUWOUTUHGZIJUEELIRBAINLUH)]
```

Key ranking by chi-square method

```
sage: ranked_cand = A.brute_force(ciphertext, ranking="chisquare")
sage: ranked_cand[:5]  # top five candidate keys
[((9, 13), AFFINECRYPTOSYSTEMGENERALIZESSHIFTCIPHER),
 ((7, 3), UTTIHOEBKRLMGKGLOCYOHOBUXIPOGGDITLEIRDOB),
 ((21, 22), HYYDUFTSVGENLVLEFBRFUFSHIDOFLLKDYETDGKFS),
 ((19, 22), FGGRSLVYPIONTPTOLXBLSLYFCRKLTTWRGOVRIWLY),
 ((5, 1), SBBWFUGHETVMOEOVUYIUFUHSRWLUOOPWBVGWTPUH)]
```

Simplified Data Encryption Standard (S-DES)

Created by Edward Schaefer [9] in 1996

Feistel round function

- Two subkeys K_1 and K_2
- Mixing function $F : (\mathbf{Z}/2\mathbf{Z})^4 \times (\mathbf{Z}/2\mathbf{Z})^8 \longrightarrow (\mathbf{Z}/2\mathbf{Z})^4$
- Feistel round function $\prod_{F,K_i} : (\mathbf{Z}/2\mathbf{Z})^8 \times (\mathbf{Z}/2\mathbf{Z})^8 \longrightarrow (\mathbf{Z}/2\mathbf{Z})^8$

Encryption and decryption

- Permutations P and P^{-1}
- Encryption function $\mathcal{E} : (\mathbf{Z}/2\mathbf{Z})^{10} \times (\mathbf{Z}/2\mathbf{Z})^8 \longrightarrow (\mathbf{Z}/2\mathbf{Z})^8$ given by $\mathcal{E} = P^{-1} \circ \prod_{F,K_2} \circ \sigma \circ \prod_{F,K_1} \circ P$
- Decryption function $\mathcal{D} : (\mathbf{Z}/2\mathbf{Z})^{10} \times (\mathbf{Z}/2\mathbf{Z})^8 \longrightarrow (\mathbf{Z}/2\mathbf{Z})^8$ given by $\mathcal{D} = P^{-1} \circ \prod_{F,K_1} \circ \sigma \circ \prod_{F,K_2} \circ P$

S-DES: Sage examples

`sage.crypto.block_cipher.sdes`

Generating subkeys

```
sage: from sage.crypto.block_cipher.sdes import SimplifiedDES
sage: sdes = SimplifiedDES()
sage: key = [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
sage: sdes.subkey(key, n=1)  # K_1
[1, 0, 1, 0, 0, 1, 0, 0]
sage: sdes.subkey(key, n=2)  # K_2
[0, 1, 0, 0, 0, 0, 1, 1]
```

Feistel round function

```
sage: block = [1, 0, 1, 1, 1, 1, 0, 1]
sage: subkey1 = sdes.subkey(key, n=1)  # K_1
sage: subkey2 = sdes.subkey(key, n=2)  # K_2
sage: sdes.permute_substitute(block, subkey1)
[0, 1, 0, 0, 1, 1, 0, 1]
sage: sdes.permute_substitute(block, subkey2)
[1, 1, 0, 0, 1, 1, 0, 1]
```

S-DES: Sage examples

Encryption & decryption

```
sage: plaintext = [0, 1, 0, 1, 0, 1, 0, 1]
sage: key = [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
sage: ciphertext = sdes.encrypt(plaintext, key)
sage: ciphertext; plaintext
[1, 1, 0, 0, 0, 0, 0, 1]
[0, 1, 0, 1, 0, 1, 0, 1]
sage: sdes.decrypt(ciphertext, key); plaintext
[0, 1, 0, 1, 0, 1, 0, 1]
[0, 1, 0, 1, 0, 1, 0, 1]
```

Longer plaintext

```
sage: bin = BinaryStrings()
sage: plaintext = bin.encoding("Encrypt this using S-DES!"); plaintext
0100010101101110011000110111001001111001011100000111010000 1... # and so on
sage: Mod(len(plaintext), 8) == 0
True
sage: key = sdes.list_to_string(sdes.random_key()); key
0010000011
sage: ciphertext = sdes(plaintext, key, algorithm="encrypt"); ciphertext
1000000110001101001110010110101101001001111100110111001010 0... # and so on
sage: sdes(ciphertext, key, algorithm="decrypt") == plaintext
True
```

Mini Advanced Encryption Standard (Mini-AES)

Created by Raphael Phan [8] in 2002

Mini-AES components

- Round keys $K_0 = K$, K_1 and K_2
- Matrix space $\mathcal{M} = \mathcal{M}_2\big(\mathbf{F}_2[x]/(x^4 + x^3 + 1)\big)$ of 2×2 matrices
- NibbleSub $\gamma : \mathcal{M} \longrightarrow \mathcal{M}$ for nibble substitution
- ShiftRow $\pi : \mathcal{M} \longrightarrow \mathcal{M}$ rotates each row
- MixColumn $\theta : \mathcal{M} \longrightarrow \mathcal{M}$ for matrix product
- KeyAddition $\sigma_{K_i} : \mathcal{M} \longrightarrow \mathcal{M}$ adds round key K_i to a matrix

Encryption & decryption

- Encryption function $\mathcal{E} : \mathcal{M} \times \mathcal{M} \longrightarrow \mathcal{M}$ given by
$$\mathcal{E} = \underbrace{\sigma_{K_2} \circ \pi \circ \gamma}_{\text{round 2}} \circ \underbrace{\sigma_{K_1} \circ \theta \circ \pi \circ \gamma}_{\text{round 1}} \circ \sigma_{K_0}$$
- Decryption function $\mathcal{D} : \mathcal{M} \times \mathcal{M} \longrightarrow \mathcal{M}$ given by
$$\mathcal{D} = \sigma_{K_0} \circ \underbrace{\gamma^{-1} \circ \pi \circ \theta \circ \sigma_{K_1}}_{\text{round 2}} \circ \underbrace{\gamma^{-1} \circ \pi \circ \sigma_{K_2}}_{\text{round 1}}$$

Mini-AES encryption

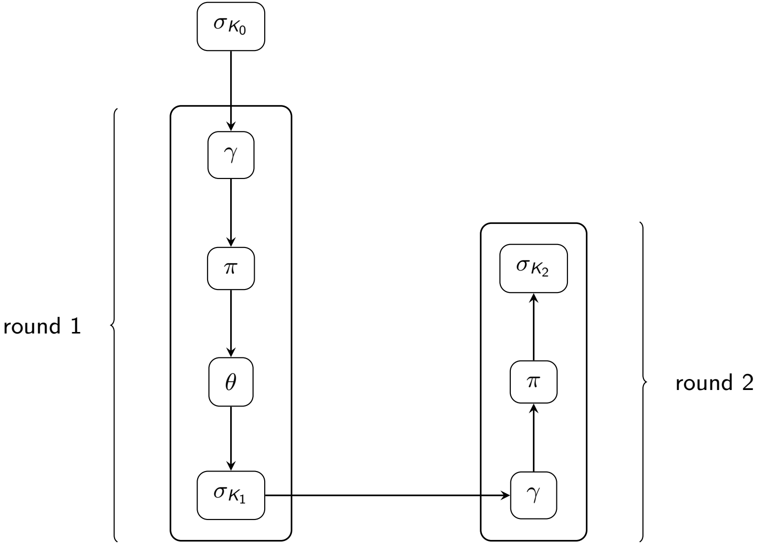

Figure 2: Two rounds in Mini-AES encryption.

Mini-AES decryption

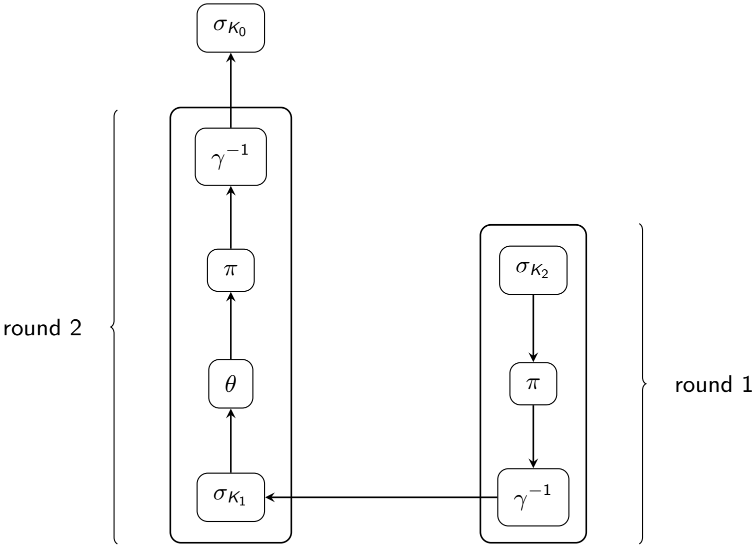

Figure 3: Two rounds in Mini-AES decryption.

Mini-AES: Sage examples

`sage.crypto.block_cipher.miniaes`

Component functions

```
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES(); maes
Mini-AES block cipher with 16-bit keys
sage: F = FiniteField(16, "x")
sage: MS = MatrixSpace(F, 2, 2)
sage: block = MS([[F("x^3 + 1"), F("x^3 + x")], [F("0"), F("x^3 + x^2")]])
sage: block
[  x^3 + 1   x^3 + x]
[        0 x^3 + x^2]
sage: key = MS([[F("x^2 + 1"), F("x^3 + x + 1")], [F("x + 1"), F("0")]])
sage: key
[    x^2 + 1 x^3 + x + 1]
[      x + 1           0]
sage: maes.add_key(block, key)     # KeyAddition
[x^3 + x^2         1]
[    x + 1 x^3 + x^2]
sage: maes.mix_column(block)       # MixColumn
[    x^3 x^2 + x]
[      1       0]
sage: maes.nibble_sub(block)       # NibbleSub
[      x^3 + x       x^2 + x]
[x^3 + x^2 + x       x^2 + 1]
sage: maes.shift_row(block)        # ShiftRow
[  x^3 + 1   x^3 + x]
[x^3 + x^2         0]
```

Mini-AES: Sage examples

Round keys

```
sage: key                          # secret key
[    x^2 + 1 x^3 + x + 1]
[      x + 1           0]
sage: maes.round_key(key, 1)       # first round key
[x^3 + x       x]
[x^3 + 1       x]
sage: maes.round_key(key, 2)       # second round key
[      x^2 + 1 x^3 + x^2 + x]
[    x^3 + x^2     x^3 + x^2]
```

Encryption & decryption

```
sage: plaintext = MS([F("x^3 + x"), F("x^2 + 1"), F("x^2 + x"), F("x^2")])
sage: plaintext
[x^3 + x x^2 + 1]
[x^2 + x     x^2]
sage: ciphertext = maes.encrypt(plaintext, key); ciphertext
[x^3 + x^2 + 1       x^3 + 1]
[            x           x^3]
sage: maes.decrypt(ciphertext, key) == plaintext
True
```

Conclusion & further work

Accomplishments

- Survey general purpose CASs for their cryptography support
- Implement missing features in Sage
- All enhancements accepted & merged in Sage (GNU GPL v2+)
- Available at `www.sagemath.org`

Further work

- Implement knapsack cryptosystems
- Wrap functionalities of PyCrypto
- Classroom use

Question & answer

Thank You

A. Baliga and S. Boztas.
Cryptography in the classroom using Maple.
In W. Yang, S. Chu, Z. Karian, and G. Fitz-Gerald, editors, *Proceedings of the Sixth Asian Technology Conference in Mathematics*, pages 343–350, 2001.

J. Cosgrave.
Number theory and cryptography (using Maple).
In D. Joyner, editor, *Coding Theory and Cryptography: From Enigma to Geheimschreiber to Quantum Theory*, pages 124–143. Springer, 2000.

M. Eisenberg.
Hill ciphers: A linear algebra project with Mathematica.
In G. Goodell, editor, *Proceedings of the Twelfth Annual International Conference on Technology in Collegiate Mathematics*, pages 100–104. Addison-Wesley, 2001.

R. E. Klima, N. P. Sigmon, and E. L. Stitzinger.
Applications of Abstract Algebra with Maple and Matlab.
Chapman & Hall/CRC, 2nd edition, 2007.

D. R. Kohel.
Cryptography, 05th November 2009.
`http://echidna.maths.usyd.edu.au/~kohel/tch/Crypto/crypto.pdf`.

M. May.
Using Maple worksheets to enable student explorations of cryptography.
Cryptologia, 33(2):151–157, 2009.

A. McAndrew.
Teaching cryptography with open-source software.

In J. D. Dougherty, S. H. Rodger, S. Fitzgerald, and M. Guzdial, editors, *SIGCSE 2008: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, pages 325–329. Association for Computing Machinery, 2008.

R. C.-W. Phan.
Mini advanced encryption standard (Mini-AES): A testbed for cryptanalysis students.
Cryptologia, 26(4):283–306, 2002.

E. F. Schaefer.
A simplified data encryption standard algorithm.
Cryptologia, 20(1):77–84, 1996.

W. Trappe and L. C. Washington.
Introduction to Cryptography with Coding Theory.
Prentice Hall, 2nd edition, 2006.