Tanda tangan Digital Menggunakan Algoritme Keccak dan RSA

Rezania Agramanisti Azdy[1]

***Abstract*—Just like conventional signatures, digital signatures can be used to prove the authenticity of a document in digital form. In addition, digital signatures can also be used to prove the validity of the author of the document, so that the author of the document can not deny that he had made the document. Digital signature is done by applying two cryptographic algorithms sequentially. The first algorithm is to impose hash function on a real document to produce message digest, and the second one is to impose a public key algorithm to the digest form. Keccak hash function is an algorithm that has been set as SHA-3 in a competition held by NIST and can be used to determine if a document has been modified or not. RSA is a public key algorithm which ensures the safety in the form of authentication and non-repudiation, so it is suitable with the actual purpose of digital signature. This paper implements the Keccak and RSA algorithm on digital signature as well as comparing it with the use of MD5. The test results of the implementation of both algorithms show that the establishment of digital signature on a whole process requires a relatively short time, and it is able to achieve the goals of digital signatures to guarantee the security in the aspects of integrity, authentication and non-repudiation.**

***Intisari*—Sama halnya seperti tanda tangan konvensional, tanda tangan digital dapat digunakan untuk membuktikan keaslian sebuah dokumen dalam bentuk digital. Selain itu, tanda tangan digital juga dapat digunakan untuk membuktikan keabsahan pembuat dokumen sehingga pembuat dokumen tidak dapat menyangkal bahwa dia yang telah membuat dokumen tersebut. Tanda tangan digital dilakukan dengan menerapkan dua algoritme kriptografi secara sekuensial. Algoritme pertama adalah memberlakukan fungsi hash pada dokumen asli untuk menghasilkan *message digest*, dan yang kedua adalah dengan memberlakukan algoritme kunci publik terhadap bentuk *digest*-nya. Keccak merupakan algoritme fungsi *hash* yang telah ditetapkan menjadi SHA-3 dari kompetisi yang diadakan oleh NIST dan dapat digunakan untuk mengetahui apakah sebuah dokumen telah dimodifikasi atau tidak. RSA adalah algoritme kunci publik yang menjamin aspek keamanan dalam bentuk autentikasi dan *non-repudiation*, sehingga sesuai dengan tujuan tanda tangan digital yang sebenarnya. Makalah ini mengimplementasikan algoritme Keccak dan RSA pada tanda tangan digital serta melihat perbandingannya dengan MD5. Hasil pengujian dari implementasi kedua algoritme di atas menunjukkan bahwa pembentukan tanda tangan digital secara keseluruhan memerlukan waktu yang relatif singkat, serta tercapai tujuan tanda tangan digital dalam menjamin keamanan pada aspek integritas, autentikasi, dan *non-repudiation*.**

Kata Kunci* — tandatangan digital, Keccak, RSA, integritas, autentikasi, *non-repudiation

[1]*Dosen, Jurusan Teknik Informatika, STMIK PalComTech, Jl. Basuki Rahmat No.05, Palembang 30129, Indonesia (tlp: 0711-359 092; fax: 0711-358 908; e-mail: rezania@palcomtech.ac.id)*

I. PENDAHULUAN

Pada dunia nyata, tanda tangan digunakan untuk meng-identifikasi keabsahan seseorang pada dokumen yang ditandatanganinya. Selain itu, tanda tangan juga dapat digunakan untuk membuktikan bahwa penandatangan telah mengetahui dan menyetujui isi dari dokumen yang ditandatanganinya tersebut. Atribut yang harus dimiliki oleh sebuah tandatangan adalah:

1. Autentikasi penandatangan, yaitu tanda tangan dapat secara unik mengidentifikasi pembuat dokumen tanpa dapat ditiru oleh orang lain.
2. Autentikasi dokumen, yaitu tanda tangan dapat digunakan untuk membuktikan keaslian dokumen, sehingga dapat diketahui jika dokumen asli telah dimodifikasi.

Dengan semakin berkembangnya internet, sebuah dokumen kini tidak hanya diterbitkan dalam bentuk cetak saja, tetapi juga dalam bentuk digital. Akan tetapi, sebuah dokumen yang ditransmisikan melalui internet sangat rentan terhadap kemungkinan modifikasi serta sulitnya pembuktian keaslian dokumen tersebut. Selain itu, seorang pengirim dapat dengan mudahnya menyangkal bahwa dialah yang telah menulis atau mengirimkan dokumen tersebut. Tanda tangan digital merupakan sebuah teknik dalam kriptografi yang dapat digunakan untuk menandatangani dokumen digital. Berbeda dengan persepsi masyarakat pada umumnya mengenai tanda tangan digital yang merupakan hasil digitalisasi dari tanda tangan seseorang, tanda tangan digital sebenarnya merupakan hasil dari diberlakukannya teknik kriptografi terhadap pesan atau dokumen asli. Tanda tangan digital harus memiliki fungsi yang sama dengan tanda tangan konvensional, yaitu dapat menjamin autentikasi, integritas, dan *non-repudiation* [1].

Perbedaan utama antara tanda tangan digital dengan tanda tangan konvensional diperlihatkan pada Tabel I. Tanda tangan konvensional dapat disalin baik secara manual maupun dengan *scan copy*, dan dapat digunakan secara berulang. Sedangkan tanda tangan digital sangat bergantung kepada dokumen yang ditandatangani, sehingga untuk setiap dokumen akan menghasilkan tandatangan yang berbeda satu sama lain.

Tanda tangan digital menggabungkan dua algoritme kriptografi sekaligus dalam implementasinya [2]. Algoritme pertama adalah menggunakan algoritme fungsi hash untuk membentuk *message digest* dari sebuah dokumen (teks), dan algoritme kunci publik yang digunakan untuk mengenkripsi *message digest* tersebut. Proses utama pada tanda tangan digital terdiri atas dua proses, yaitu proses *signing* (tanda tangan) dan verifikasi. Proses *signing* dilakukan dengan mengubah sebuah pesan atau dokumen menjadi *message digest*-nya, dan mengenkripsinya menggunakan algoritme kunci publik. Sementara, proses verifikasi dilakukan dengan

membandingkan hasil dekripsi pesan yang diterima (*ciphertext*) dengan *message digest* dari pesan sebenarnya.

TABEL I
PERBANDINGAN ANTARA TANDA TANGAN KONVENSIONAL DENGAN TANDA TANGAN DIGITAL

	Tandatangan Konvensional	Tandatangan Digital
Bentuk		A782D56BWF955761 C7CD892636AF9892 98BC678299BDA431 176446787632145760
Masalah	Dapat digunakan kembali.	Tidak dapat digunakan kembali.
	Tandatangan terletak dibagian paling belakang dokumen.	Tandatangan diaplikasikan ke seluruh dokumen.

Algoritme fungsi hash yang umum digunakan adalah MD5 [3], akan tetapi seiring dengan ditemukannya serangan-serangan terhadap MD5 khususnya *collision attack* [4], maka keamanan MD5 untuk tanda tangan digital tidak lagi terjamin. Hal ini dikarenakan adanya kemungkinan untuk menemukan dua buah pesan atau dokumen yang berbeda dan menghasilkan nilai hash yang sama, sehingga aspek integritas yang harus dimiliki oleh algoritme fungsi hash tidak dapat terpenuhi.

Keccak merupakan salah satu algoritme fungsi *hash* yang dirancang oleh Guido Bertoni, Joan Daemen, Michaël Peeteres, dan Gilles Van Assche [5]. Keccak merupakan pemenang kompetisi *SHA-3 Cryptographic* Hash *Algorithm Competition* yang diselenggarakan oleh NIST dan telah dijadikan standar untuk algoritme fungsi hash *Secure* Hash *Algorithm* (SHA-3) yang baru [6]. Terpilihnya Keccak sebagai pemenang SHA-3 menjadikan Keccak sebagai algoritme fungsi hash standar yang telah terbukti keamanannya. Perbedaan utama antara Keccak dengan SHA-1, SHA-2, ataupun MD5 adalah Keccak menggunakan konstruksi spons *function* pada pembentukan nilai hash-nya, sedangkan tiga algoritme tersebut menggunakan skema Merkle-Damgård.

RSA merupakan sebuah *cryptosystem* yang dikemukakan oleh Ron Rivest, Adi Shamir, dan Len Adleman pertama kalinya pada tahun 1977 [7]. RSA digunakan untuk menjamin *privacy* dan keaslian data digital [8]. Misalnya, RSA digunakan untuk mengamankan lalu lintas *web*, *email*, sesi *remote login*, dan sistem pembayaran kartu kredit elektronik [9]. Keamanan RSA terletak pada sulitnya memfaktorkan bilangan prima yang sangat besar menjadi faktor-faktor prima yang lebih kecil. Pemfaktoran ini dilakukan untuk memperoleh kunci privat yang dapat digunakan untuk memperoleh pesan asli (*plaintext*). RSA digunakan untuk proses enkripsi dan dekripsi pesan, dan juga memiliki kemampuan untuk menandatangani dan verifikasi paket data. Selain itu RSA tidak menentukan penggunaan algoritme tertentu untuk fungsi hash-nya, sehingga keamanan untuk proses enkripsi dan tanda tangan tidak bergantung pada algoritme fungsi hash-nya [10].

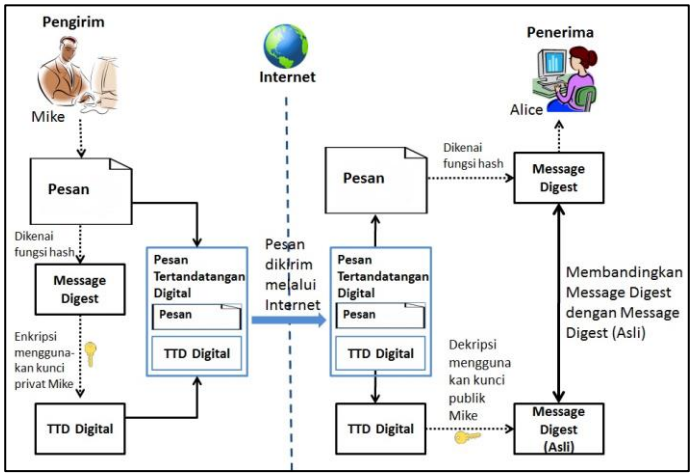
Dalam makalah ini akan dibahas implementasi algoritme Keccak dan RSA pada tanda tangan digital serta perbandingannya dengan algoritme fungsi hash yang umum digunakan, yaitu MD5. Implementasi diharapkan dapat memberikan hasil yang maksimal, mulai dari cepatnya proses tanda tangan, verifikasi, dan pembangkitan kunci yang aman pada RSA, serta dapat menjamin aspek keamanan yang dimiliki oleh tanda tangan digital.

Pembahasan makalah ini akan dimulai dengan definisi dan uraian tentang tanda tangan digital. Bab III memberikan gambaran alur kerja algoritme Keccak dan RSA. Bab IV memberikan pembahasan implementasi Keccak dan RSA pada tanda tangan digital. Bab V merupakan pemaparan hasil dan analisis dari implementasi algoritme, dan bab terakhir merupakan kesimpulan dari makalah.

II. SKEMA TANDA TANGAN DIGITAL

Tanda tangan digital pertama kali diperkenalkan oleh Diffie and Hellman pada tahun 1976 [11]. Tanda tangan digital menggunakan *cryptosystem* kunci publik, yaitu sebuah metode kriptografi yang menggunakan sepasang kunci yang asimetris, yaitu kunci publik dan kunci privat. Kunci publik dapat disebarluaskan dan kunci privat tidak boleh didistribusikan dan harus dirahasiakan. Dengan menggunakan pasangan kunci asimetris ini, maka data yang dienkripsi menggunakan sebuah kunci publik hanya dapat didekripsi menggunakan kunci privatnya. Berlaku juga sebaliknya.

Gbr. 1 memperlihatkan cara kerja tanda tangan digital. Mike menuliskan sebuah dokumen dan hendak mengirimkannya kepada Alice. Dengan menggunakan fungsi hash, Mike mengubah dokumen yang telah ditulisnya menjadi sebuah *message digest* dan kemudian mengenkripsi *message digest* tersebut menggunakan kunci privat miliknya. Hasil enkripsi *message digest* inilah yang disebut dengan tanda tangan digital. Tanda tangan digital ini kemudian dapat di-*attach* ke dokumen asli atau dikirim secara terpisah tetapi pada waktu yang bersamaan, lalu dikirimkan kepada Alice melalui internet.



Gbr. 1 Contoh tanda tangan digital.

Alice menerima pesan yang dikirim oleh Mike beserta tanda tangan digitalnya. Alice membagi kedua pesan tersebut menjadi dokumen asli dan tanda tangan digitalnya. Alice mengenakan fungsi hash dari dokumen asli dan memperoleh

message digest dari dokumen tersebut. Alice melakukan dekripsi terhadap tanda tangan digital dengan menggunakan kunci publik milik Mike dan memperoleh *plaintext*-nya, kemudian membandingkan *plaintext* tersebut dengan *message digest* yang diperolehnya. Jika hasilnya sama, maka dapat diperoleh kesimpulan berupa:

1. Pesan tersebut benar ditulis oleh Mike karena hanya Mike yang mengetahui kunci privat miliknya.
2. Pesan tersebut tidak dimodifikasi oleh orang lain.

Segala aktivitas yang dilakukan Mike disebut dengan proses *signing* atau penandatanganan, dan aktivitas yang dilakukan Alice adalah proses verifikasi.

Tanda tangan digital dapat menjamin keamanan dokumen yang ditandatanganinya dalam aspek *integrity*, *authentication*, serta *non-repudiation* [2], [12].

1) Integrity: Tanda tangan digital dapat memastikan bahwa pesan yang dikirim adalah pesan yang sebenarnya dan tidak dimodifikasi pada saat transmisi. Hal ini dikarenakan pada dokumen asli tidak diberlakukan proses enkripsi, sehingga dokumen dapat dibaca oleh banyak pihak. Keaslian pesan dapat diperoleh dengan membandingkan *message digest* dari dokumen asli dengan *plaintext* hasil verifikasi tandatangan digital. Jika hasilnya sama, maka dokumen telah terjamin keasliannya.

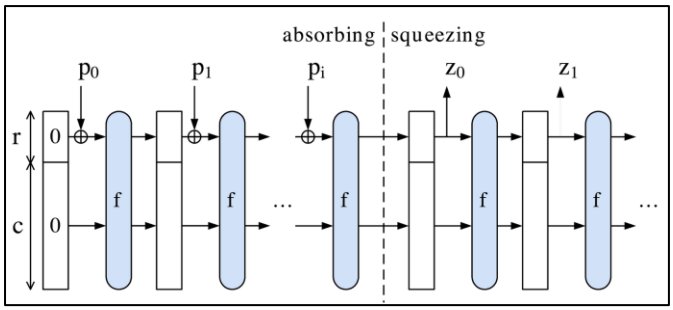
2) Authenticity: Tanda tangan digital dibuat dengan mengenkripsi *message digest* dari pesan asli menggunakan kunci privat pengirim. *Ciphertext* ini hanya bisa didekripsi menggunakan pasangan kunci publik dari kunci privat tersebut, sehingga jika hasil verifikasi membuktikan bahwa *message digest* dokumen sama dengan hasil dekripsi dari *ciphertext*, maka dapat dipastikan bahwa pengirim adalah benar orang yang memiliki kunci privat tersebut.

3) Non-repudiation: Jika tanda tangan digital telah terbukti ditandatangani menggunakan kunci privat tertentu, maka tidak dapat disangkal bahwa pemilik kunci privatlah yang telah menulis dokumen tersebut.

III. ALGORITME YANG DIGUNAKAN PADA TANDA TANGAN DIGITAL

A. Keccak

1) Konstruksi Spons: Keccak mengadopsi *sponge structure* [13] yang melakukan permutasi terhadap *state* dengan panjang yang tetap (*b*).



Gbr. 2 Ilustrasi konstruksi spons [6].

Gbr. 2 memperlihatkan fase-fase yang ada pada konstruksi spons, yaitu fase *absorbing* dan fase *squeezing*. Analoginya terhadap sebuah spons yaitu fungsi f "absorbs" *arbitrary input bit* ke dalam *state*-nya, dan "squeezes" *arbitrary output bit* dari *state*-nya. Konstruksi spons menggunakan tiga buah komponen berikut [6]:

1. Sebuah fungsi yang mendasari pada string dengan panjang yang tetap (dinotasikan dengan *f*).
2. Sebuah parameter yang disebut dengan *rate* (dinotasikan dengan *r*).
3. Sebuah aturan *padding* (dinotasikan dengan *pad*).

Fungsi yang dibangun menggunakan komponen tersebut, dinotasikan dengan SPONGE[*f*, *pad*, *r*], disebut dengan *sponge function* [6].

Aturan *padding* merupakan sebuah fungsi yang menghasilkan nilai *padding*, yaitu sebuah *string* dengan panjang tertentu untuk ditambahkan pada *string* lainnya. Keccak menggunakan prinsip *multi-rate padding*, yang dinotasikan dengan 10*1, yaitu menambahkan bit 1 diikuti dengan sejumlah bit 0 hingga diakhiri dengan bit 1 kembali. Aturan *padding* dikenakan terhadap *input* berupa pesan *M*, guna membuat panjang *M* menjadi kelipatan dari *r* (*bitrate*).

Rate r adalah bilangan bulat positif yang panjangnya kurang dari panjang *b*. *Capacity c*, adalah selisih dari *r* dengan *b* sehingga:

$$r + c = b \tag{1}$$

Fungsi *f* merupakan fungsi permutasi yang memetakan string dengan panjang yang tetap (panjang *string* dinotasikan dengan *b*) ke *string* dengan panjang yang sama.

2) KECCAK-p: Merupakan sebuah permutasi yang ditentukan oleh dua buah parameter [6]:

1. Panjang tetap dari *string* yang akan dipermutasi (disebut dengan lebar permutasi) – dinotasikan dengan *b*.
2. Banyaknya iterasi dari transformasi internal (disebut dengan *round*) – banyaknya *round* dinotasikan dengan n_r.

Permutasi KECCAK-*p* sebanyak n_r putaran dengan lebar *b* dinotasikan dengan KECCAK-*p* [*b*, n_r]. Permutasi didefinisikan untuk setiap $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ dan sembarang bilangan bulat positif n_r. *Input* b pada setiap putaran yang dikenakan pada permutasi akan menghasilkan sebuah *state* yang akan digunakan sebagai masukan pada putaran berikutnya. Panjang dari *state* sama dengan panjang *b*, yaitu:

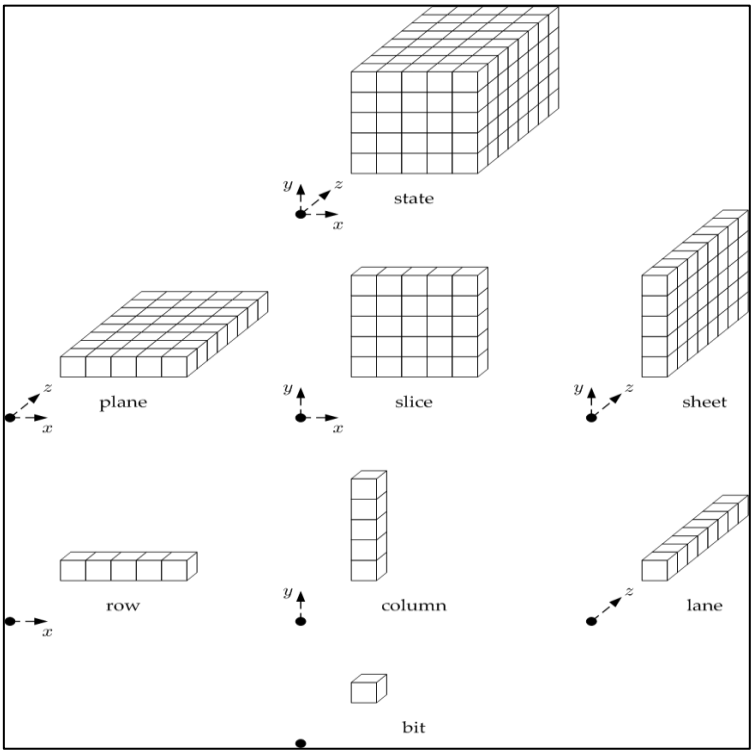
$$b = 5 \text{ x } 5 \text{ x } w \tag{2}$$

dengan $w = 2^l$ dan $l = 0, 1, \ldots, 6$.

Jika *S* melambangkan *string* yang mewakili sebuah *state*, maka bit-bitnya diindeks mulai dari 0 hingga *b*-1, sehingga:

$$S = S[0] \,||\, S[1] \,||\, S[2] \,||\, \ldots \,||\, S[b-1] \tag{3}$$

Dan jika *A* melambangkan *state* yang berupa *bit array*, maka bit-bitnya ditunjukkan oleh indeks (*x*, *y*, *z*) dengan $0 \le x < 5$, $0 \le y < 5$, dan $0 \le z < 2^l$. Pada umumnya, sebuah *state* direpresentasikan oleh *array* tiga dimensi, A[*x*][*y*][*z*].



Gbr. 3 Ilustrasi *state* pada Keccak [6].

Gbr. 3 memperlihatkan ilustrasi *state* yang terjadi saat pemrosesan pada Keccak. Setiap putaran pada permutasi KECCAK-*p* dinotasikan dengan Rnd, terdiri atas serangkaian lima buah operasi yang disebut dengan *Step Mappings*. *Input* dari setiap proses merupakan *state array* A, dan *output* yang dihasilkan juga berupa *state array* (dinotasikan dengan A') yang akan menjadi *input* untuk putaran berikutnya. Operasi pada *Step Mappings* ini meliputi:

1. θ (theta-*function*)
   Fungsi ini akan melakukan operasi XOR antara setiap bit pada *state* dengan paritas dari dua buah kolom pada *array*.
   Algoritme θ-function: (*input*: *state array A; output*: *state array A'*)
   i. Untuk semua pasangan (*x*, *z*) dengan 0≤*x*<5 dan 0≤*z*<*w*, maka:
   $$C[x,z] = A[x,0,z] \oplus A[x,1,z] \oplus A[x,2,z] \oplus A[x,3,z] \oplus A[x,4,z] \quad (4)$$
   ii. Untuk semua pasangan (*x*, *z*) dengan 0≤x<5 dan 0≤z<w, maka:
   $$D[x,z] = C[(x-1) mod\ 5, z] \oplus C[(x+1) mod\ 5, (z-1) mod\ w] \quad (5)$$
   iii. Untuk semua *triple* (*x*, *y*, *z*) dengan 0≤ x< 5, 0≤y<5, dan 0≤z<w, maka:
   $$A'[x,y,z] = A[x,y,z] \oplus D[x,z] \quad (6)$$
2. ρ (rho-*function*)
   Fungsi ini akan merotasikan bit-bit pada setiap *lane* dengan panjang yang bergantung pada koordinat *x* dan *y* pada *lane* (disebut *offset*). Dengan kata lain, untuk setiap bit pada *lane*, koordinat *z* dimodifikasi menambahkan *offset*-nya di modulo dengan ukuran *lane*.
   Algoritme ρ-*function*: (*input*: *state array A; output*: *state array A'*)
   i. Untuk semua z dengan 0≤z<w, maka:
   $$A'[0,0,z] = A[0,0,z] \quad (7)$$
   ii. (*x*, *y*) = (1, 0)
   iii. Untuk *t* dari 0 hingga 23:
      a. Untuk semua *z* dengan 0≤*z*<*w*, maka:
      $$A'[x,y,z] = A[x,y, \left(z - \frac{(t+1)(t+2)}{2}\right) 2\ mod\ w] \quad (8)$$
      b. (x, y) = (y, (2x+3y) mod 5)
   iv. Return *A'*
3. π (pi-*function*)
   Inti dari fungsi ini adalah transposisi terhadap sebuah *lane*, yaitu mengatur ulang posisi dari *lane*.
   Algoritme π-*function*: (*input*: *state array A; output*: *state array A'*)
   i. Untuk semua *triple* (*x*, *y*, *z*) dengan 0 ≤ *x* < 5, 0 ≤ *y* < 5, dan 0 ≤ *z* < *w*, maka:
   $$A'[x,y,z] = A[(x+3y) mod\ 5, x, z] \quad (9)$$
   ii. Return *A'*
4. χ (chi-*function*)
   Fungsi akan melakukan operasi XOR dengan sebuah fungsi non-linear antara setiap bit dengan dua bit lainnya pada setiap *row*.

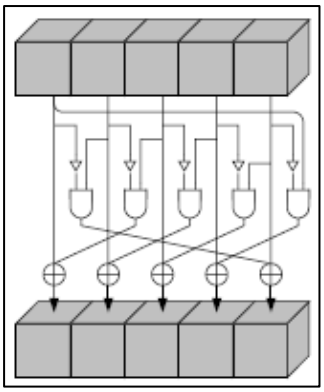
Algoritme χ-*function*: (*input*: *state array A; output*: *state array A'*)

i. Untuk semua *triple* (*x*, *y*, *z*) dengan 0≤*x*<5, 0≤*y*<5, dan 0≤*z*<*w*, maka:

$$A'[x,y,z] = A[x,y,z] \oplus ((A[(x+1) mod\ 5, y, z] \oplus 1) AND\ A[(x+2) mod\ 5, y, z]) \quad (10)$$

ii. Return *A'*

Gbr. 4 memperlihatkan ilustrasi χ-*function* yang diberlakukan terhadap sebuah *single row*. Operasi ini merupakan operasi *mapping* non-linear satu-satunya pada Keccak.



Gbr. 4 χ-*function* diterapkan pada setiap baris [6].

5. ι (iota-*function*)

Fungsi ini memerlukan parameter berupa *round index* (dinotasikan dengan i_r). Fungsi ini memerlukan perhitungan dari fungsi lain (RC) yang berdasarkan pada *linear feedback shift register*.

Algoritme fungsi RC: (*input*: *integer i*; *output*: bit *rc(t)*)

i. if *t* mod 255=0, return 1.
ii. R = 10000000.
iii. Untuk *i* dari 1 hingga *t* mod 255:
   a. *R = 0 || R;*
   b. *R[0] = R[0] + R[8];*
   c. *R[4] = R[4] + R[8];*
   d. *R[5] = R[5] + R[8];*
   e. *R[6] = R[6] + R[8];*
   f. *R = Trunc*$_8$*[R].*
iv. Return *R[0]*.

Algoritme ι-*function*: (*input*: *state array A*, *round index* i_r*; output*: *state array A'*)

i. Untuk semua *triple* (*x*, *y*, *z*) dengan 0≤*x*<5, 0≤*y*<5, dan 0≤*z*<*w*, maka *A'[x, y, z] = A[x, y, z].*
ii. *RC = 0*^{w}.
iii. Untuk *j* dari 1 hingga *l*, maka:

$$RC[2^j-1] = rc[j+7i_r]. \quad (11)$$

iv. Untuk semua z dengan 0 ≤ z < w, maka:

$$A'[0,0,z] = A'[0,0,z] \oplus RC[z] \quad (12)$$

v. Return *A'*.

B. RSA

RSA merupakan sebuah algoritme yang meng-implementasikan kriptosistem kunci publik dengan ide utama berupa [7]:

1. *Public key encryption.*
   Tidak diperlukan jalur transmisi yang aman untuk mendistribusikan kunci. Setiap orang memiliki sepasang kunci berupa kunci publik dan kunci privat. Kunci publik disebarluaskan secara umum dan digunakan untuk mengenkripsi pesan yang akan dikirimkan kepada pemiliknya. Meskipun pesan dikirim melalui jalur yang tidak aman, pesan tetap terjaga keutuhannya karena hanya pemilik kunci yang dapat mendekripnya.
2. *Signature.*
   Penerima pesan dapat memverifikasi apakah pesan yang diterimanya benar-benar dikirim si pengirim. Tanda tangan dibuat menggunakan kunci privat milik pengirim, dan siapapun penerimanya dapat memverifikasinya meng-gunakan kunci publik pengirim.

RSA merupakan algoritme yang aman dikarenakan sulitnya memfaktorkan bilangan *n*, di mana *n = pq*, *p* dan *q* merupakan bilangan prima yang sangat besar [14].

1) Algoritme Pembangkitan Kunci: Algoritme pem-bangkitan kunci pada RSA adalah sebagai berikut:

1. Membangkitkan dua buah bilangan prima sembarang, *p* dan *q*.
2. Menghitung *n*, yaitu:

$$n = pq \quad (13)$$

3. Menghitung $\phi(n)$, yaitu:

$$\phi(n) = (p-1)(q-1) \quad (14)$$

4. Memilih kunci publik *e* yang relatif prima terhadap $\phi(n)$ (gcd($\phi(n), e) = 1$)
5. Membangkitkan kunci privat *d* menggunakan (15)

$$e.d = k.\phi(n) + 1 \quad (15)$$

Hasil dari algoritme tersebut adalah kunci publik (pasangan *e* dan *n*) dan kunci privat (pasangan *d* dan *n*).

2) Algoritme Enkripsi dan Dekripsi: Algoritme enkripsi pada RSA adalah sebagai berikut:

1. Mengambil kunci publik penerima pesan (pasangan *e* dan *n*).
2. Membagi pesan M menjadi blok-blok m_1, m_2, …, m_i; dengan syarat setiap blok nilainya masih dalam interval [*0*, *n-1*].
3. Setiap blok m_i dienkripsi menjadi c_i menggunakan (16):

$$c_i = {m_i}^e \ mod\ n \quad (16)$$

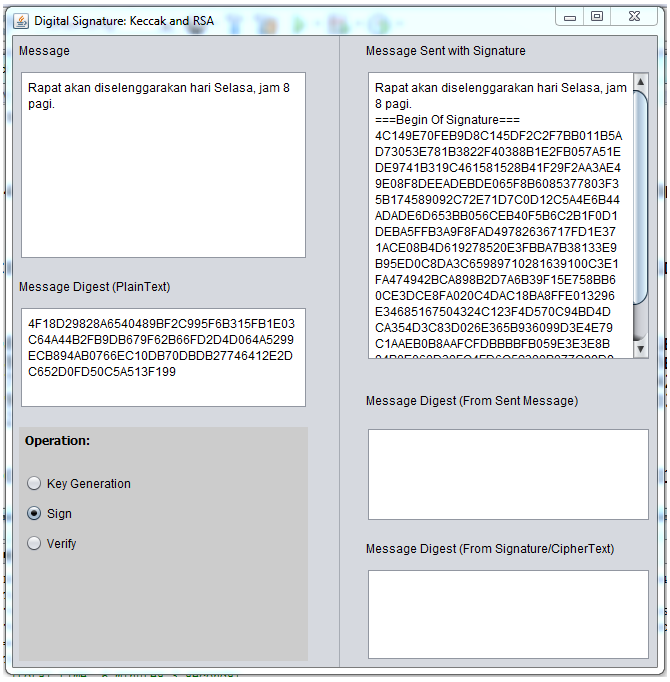
Proses dekripsi memiliki algoritme yang sama dengan proses enkripsinya, hanya berbeda pada langkah ke-3, yaitu setiap blok c_i didekripsi menjadi m_i menggunakan (17):

$$m_i = {c_i}^d \ mod\ n \quad (17)$$

IV. IMPLEMENTASI

Dalam mengimplementasikan algoritme Keccak dan RSA, pada makalah ini digunakan program sederhana untuk menunjukkan cara kerja tanda tangan digital. Program ditulis menggunakan bahasa Java dan memanfaatkan API Bouncy Castle sebagai *provider* yang mendukung kriptografi pada Java.

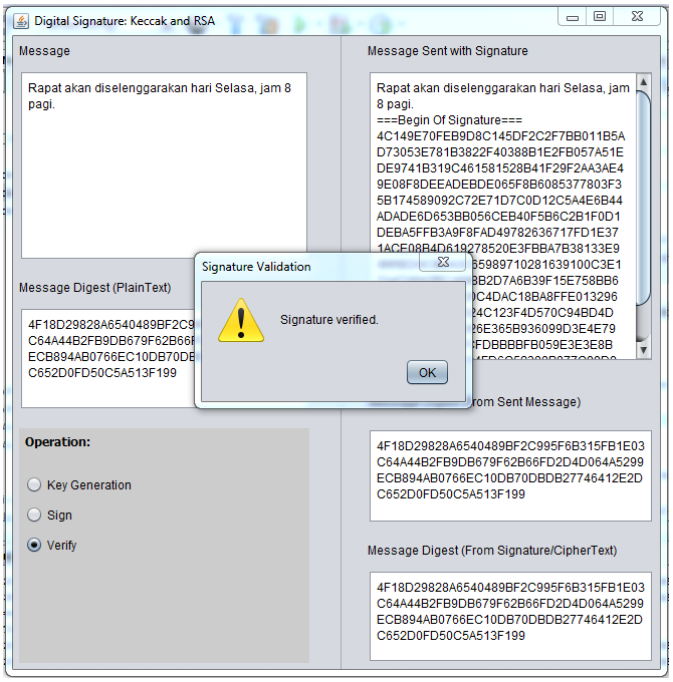
Program berfungsi dengan mengambil *input* pesan secara langsung dari *field* yang disediakan. Terdapat tiga operasi yang dapat dilakukan. Operasi yang pertama adalah *Key Generation* untuk membangkitkan kunci. Operasi yang kedua adalah *Sign* untuk mengubah pesan asli ke dalam bentuk *digest*, lalu mengenkripnya dan menampilkan hasilnya dalam bentuk heksadesimal ke dalam *field Signature*. Pada tanda tangan digital, enkripsi *message digest* dari pesan menggunakan algoritme RSA dilakukan menggunakan kunci privat pengirim sehingga dapat digunakan untuk menjamin identitas pengirim. Gbr. 5 memperlihatkan tampilan program sederhana, di mana *input* pesan asli dapat diketikkan secara langsung pada *field message*.



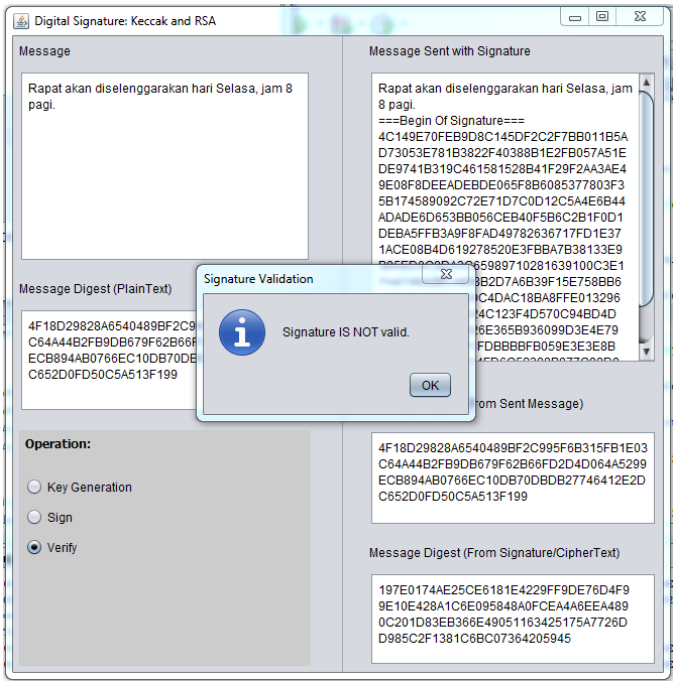
Gbr. 5 Tampilan program.

Operasi yang terakhir adalah *Verify,* yang digunakan untuk memeriksa keabsahan tanda tangan digital. Operasi ini akan memisahkan pesan yang dikirim dengan tanda tangan digital dari pesan yang dikirim tersebut. Fungsi hash dikenakan pada pesan yang dikirim agar diperoleh bentuk *digest*-nya, dan proses dekripsi dikenakan pada tanda tangan digitalnya agar diperoleh bentuk *digest* dari pesan asli yang dikirim dan belum diubah oleh *intruder*. Dekripsi dilakukan dengan menggunakan kunci publik milik pengirim sehingga dapat menghasilkan *plaintext* yang sama dengan aslinya.

Gbr. 6 memperlihatkan hasil verifikasi ketika pesan terkirim dalam keadaan asli dan tidak diubah, dan Gbr. 7 memperlihatkan hasil verifikasi baik ketika pesan telah diubah saat dikirim atau tanda tangan digital didekripsi menggunakan kunci publik yang bukan pasangannya.



Gbr. 6 Hasil verifikasi tanda tangan digital valid.



Gbr. 7 Hasil verifikasi tanda tangan digital tidak valid.

V. HASIL DAN PEMBAHASAN

Pengujian dilakukan dengan menggunakan metode *blackbox testing* untuk memastikan program berjalan sesuai dengan yang diharapkan. Pengujian dilakukan dengan menghitung waktu yang diperlukan untuk proses membangkitkan kunci, tanda tangan, dan verifikasi. Pengujian

dilakukan dengan mengeksekusi program sebanyak sepuluh kali, dan hasil eksekusi program diperlihatkan pada Tabel II.

TABEL II
PERBANDINGAN HASIL EKSEKUSI PENGUJIAN LAMANYA WAKTU PROSES TANDA TANGAN

Percobaan Ke-	Waktu			Hasil Verifikasi
	Pembangkitan Kunci	Tanda tangan	Verifikasi	
1	1,704	597	2,010	Valid
2	1,006	611	1,976	Valid
3	1,618	588	2,188	Valid
4	1,675	579	1,770	Tidak Valid
5	1,983	544	1,613	Tidak Valid
6	1,043	594	1,242	Tidak Valid
7	1,472	632	2,067	Valid
8	1,861	591	1,807	Tidak Valid
9	696	564	1,570	Valid
10	1,169	635	1,628	Tidak Valid
Rata-rata	**1,422.70**	**593.50**	**1,787.10**	

Dari hasil pengujian dengan ukuran dokumen yang sama diperoleh waktu yang relatif singkat untuk proses tanda tangan digital secara keseluruhan. Kunci yang dibangkitkan berukuran 2048 bit. Dan hasil verifikasi yang tidak valid memerlukan waktu verifikasi lebih singkat daripada hasil verifikasi yang valid.

Selain pengujian di atas, pengujian dilakukan dengan melakukan analisis terhadap beberapa komponen.

A. Analisa Integrity

Pengujian dilakukan dengan melakukan proses tanda tangan dan verifikasi tanpa memodifikasi pesan asli. Program memberikan hasil tanda tangan terverifikasi yang menunjukkan bahwa pesan yang diterima sama dengan pesan asli, sehingga pesan dapat dipertanggungjawabkan keasliannya.

Pesan asli: Rapat akan diselenggarakan hari Selasa, jam 8 pagi.

***Digest* pesan asli (dalam bentuk heksadesimal):**
4F18D29828A6540489BF2C995F6B315FB1E03C64A44
B2FB9DB679F62B66FD2D4D064A5299ECB894AB076
6EC10DB70DBDB27746412E2DC652D0FD50C5A513F
199

Tandatangan digital (dalam bentuk heksadesimal):
97473523C531F706E66E6DF02D7C2CA0247AB1E52C1
0845FEF6A14F782709B878172C513ACF5FA5EEAA76
23FF3BD5E17AD4F6EF433B51B68E5FB31EC57E181E
B5EE41EFE98C14436FE0968C190F8FFAF93720D00D7
D6436A5E9250CBD24E25CCC433D2D89F2A7B4A232
DC871C4EE6732AC69330B608DB156E05E5B7AB6B66
C8F56CAD1171D2C5C87A46BFB49DE0FBD5E6B7DD
590E9A3BF86CCF2A0F9733A43D9D635F2B33EF6D97
08DB771A84BBD2099A4A554A9F3F5C3198F23985E7
1D95912634D6CE75D7253CA68ABE94BE8342C5F466
7386BB68F613DA2061792A912A7DEC70BAA45AEB4
C7D9B053FD819EEFCE8500CF3B422A76C8B397B755
5D05E86631

***Digest* pesan asli (hasil dekripsi dari tandatangan digital – dalam bentuk heksadesimal):**
4F18D29828A6540489BF2C995F6B315FB1E03C64A44
B2FB9DB679F62B66FD2D4D064A5299ECB894AB076
6EC10DB70DBDB27746412E2DC652D0FD50C5A513F
199

Pengujian terhadap keaslian pesan juga dilakukan dengan melakukan modifikasi terhadap pesan asli yang dikirim.

Pesan asli termodifikasi: Rapat akan diselenggarakan hari Selasa, jam 9 pagi.

***Digest* pesan termodifikasi (dalam bentuk heksadesimal):**
197E0174AE25CE6181E4229FF9DE76D4F99E10E428A
1C6E095848A0FCEA4A6EEA4890C201D83EB366E490
51163425175A7726DD985C2F1381C6BC07364205945

***Digest* pesan asli (hasil dekripsi dari tandatangan digital – dalam bentuk heksadesimal):**
4F18D29828A6540489BF2C995F6B315FB1E03C64A44
B2FB9DB679F62B66FD2D4D064A5299ECB894AB076
6EC10DB70DBDB27746412E2DC652D0FD50C5A513F
199

Dengan membandingkan *digest* yang dibentuk dari pesan yang dikirim (dan telah dimodifikasi) dengan *digest* hasil dekripsi dari tanda tangan digital, diperoleh hasil yang berbeda, sehingga dapat diketahui bahwa pesan yang dikirim telah dimodifikasi dalam perjalanannya menuju ke penerima, dan dapat membuktikan asli atau tidaknya data yang diterima.

B. Analisis Authenticity

Pengujian *authenticity* digunakan untuk memeriksa keabsahan pengirim pesan. Pengujian dilandasi dengan penggunaan kriptografi asimetri yang dimiliki oleh RSA. Satu pasang kunci yang dibangkitkan hanya dapat digunakan untuk mengenkrip/mendekrip pesan/*ciphertext* yang diproses dengan kunci pasangannya, sehingga tidak mungkin diperoleh sebuah *plaintext* yang sama persis dengan pesan asli jika proses dekripsi dilakukan dengan menggunakan kunci publik yang bukan pasangannya.

Pengujian pada program dilakukan dengan membangkitkan dua pasangan kunci, yaitu pasangan kunci A dan pasangan kunci B. Pengujian pertama dilakukan dengan mengenkripsi *message digest* menggunakan kunci privat A, dan mendekripnya menggunakan kunci publik B. Hasil dari pengujian menampilkan pesan bahwa tanda tangan terverifikasi, yang berarti tanda tangan tersebut sah dibuat oleh pemilik kunci privat A. Pengujian kedua dilakukan dengan menggunakan kunci privat A, dan mendekrip *ciphertext*-nya dengan menggunakan kunci publik B. Hasil dari pengujian menampilkan bahwa tanda tangan tidak valid. Hal ini dikarenakan proses dekripsi yang dilakukan dengan menggunakan kunci privat yang berbeda akan menghasilkan *plaintext* yang berbeda pula, sehingga perbandingan *message*

digest pesan asli dengan *message digest* pesan yang terkirim (hasil dekripsi dari *ciphertext*) tidaklah sama.

C. Analisis Non-repudiation

RSA merupakan algoritme asimetri yang menggunakan kunci yang berbeda untuk proses enkripsi dan dekripsinya. Kunci yang dibangkitkan merupakan sepasang kunci, yaitu kunci publik yang dapat disebarluaskan, dan kunci privat yang sifatnya rahasia. Berbeda dengan skema algoritme asimetri lainnya, pada tanda tangan digital yang digunakan untuk proses enkripsi adalah kunci privatnya. Hal ini dikarenakan salah tujuan dari tanda tangan digital adalah berkemampuan untuk memastikan keabsahan pembuat dokumen. Ketika pembuat dokumen menandatangani secara digital dengan menggunakan kunci privatnya, dan penerima dapat memverifikasi tanda tangan digital tersebut dengan kunci publik yang berpasangan dengan kunci privat tersebut, maka dapat dipastikan keabsahan pembuat dokumen tersebut, sehingga pembuat dokumen pun tidak dapat menyangkal bahwa dialah yang membuat dokumen tersebut.

D. Perbandingan dengan MD5

Algoritme MD5 mengolah bit pesan dengan panjang *state internal*-nya 4x32 bit dan menghasilkan nilai hash 128 bit untuk berapapun panjang *input* pesan yang diolah. Pada Keccak, panjang nilai hash dapat berupa 224, 256, 384, atau 512 bit dengan panjang *state* internal 5x5x64 bit untuk standar SHA-3. Letak keamanan pada Keccak terletak pada nilai *c* karena nilai ini tidak terpengaruh baik oleh *input* maupun oleh *output* (tidak pernah dikeluarkan pada proses *squeezing*) dan tahan terhadap serangan hingga kompleksitas $2^{c/2}$.

Waktu eksekusi pembentukan *message digest* menggunakan Keccak tidak tergantung pada *input* yang diprosesnya. Desain dari permutasi Keccak mengikuti prinsip Matryoshka *principle* yang menjadikan keamanan dari tujuh buah permutasi pada Keccak saling terhubung satu sama lain. Kriptoanalis pada f[x] dengan x yang lebih rendah akan berpengaruh pada permutasi berikutnya dengan nilai x yang lebih tinggi.

Padding pada MD5 dapat menghasilkan *security proof*, karena *padding* pada MD5 dilakukan dengan menambahkan angka 1 yang diikuti dengan beberapa angka 0 dan diakhiri dengan panjang ukuran *input* pesan pada akhir bit *padding*. Sedangkan Keccak yang menggunakan struktur spons untuk skema konstruksinya relatif lebih aman untuk proses kompresi pada fungsi permutasinya.

VI. KESIMPULAN

Pada makalah ini telah diimplementasikan algoritme Keccak dan RSA pada tanda tangan digital dengan menggunakan program sederhana yang ditulis dengan bahasa Java. Program dibangun menggunakan *provider* Bouncy Castle yang memudahkan penulis dalam mengimplemetasikan algoritme kriptografi pada Java.

Penggunaan algoritme Keccak berhasil menjaga aspek keamanan dalam hal integritas data. Hal ini dikarenakan sampai saat ini Keccak masih bersifat *collision resistance*, sehingga belum ditemukan adanya dua data yang berbeda menghasilkan *message digest* yang sama. Penggunaan algoritme RSA dapat menjamin aspek keamanan dalam hal autentikasi dan *non-repudiation*. Pembangkitan kunci pada RSA memastikan hanya pasangan kunci yang digunakan untuk proses enkripsilah yang dapat digunakan untuk proses dekripsinya. Sehingga dengan pasangan kunci yang tepat proses enkripsi dan dekripsi dapat menghasilkan hasil yang tepat, dan tidak dapat dielakkan lagi bahwa pembuat dokumen adalah orang yang sama dengan pemilik kunci yang digunakan untuk proses enkripsinya.

Pengujian penelitian ini memberikan hasil bahwa implementasi algoritme Keccak dan RSA pada tanda tangan digital dapat menjamin keaslian dokumen yang diterima, keabsahan pembuatan dokumen, dan anti penyangkalan oleh pembuat dokumen.

REFERENSI

[1] Y. F. Lim, "Digital signatures, certification authorities: certainty in the allocation of liability", *Singapore Journal of International & Comparative Law*, vol. 7, issue 1 , pp. 183-200, 2003.

[2] C. Romine, "Digital Signature Standard (DSS)", Federal Information Processing Standards Publication, Information Technology Laboratory National Institute of Standards and Technology, Jul. 2013.

[3] Anyapu, S., Aparna, G., Manognya, R., Kumar, D.R., "Message Security Through Digital Signature Generation and Message Digest Algorithm", *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 3, pp.300-304, Mar. 2013.

[4] Stevens, M.M.J., "On Collisions for MD5", Master's Thesis, Dept. of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, Jun. 2007.

[5] C. Bentivenga, F. Christie, and M. Kitson. (2010) Keccak Final Paper. [Online]. Available: https://www.cs.rit.edu/~ark/winter2012/482/team/u5/report.pdf

[6] C. H. Romine. (Agt. 2015) SHA-3 Standard: Permutation-Based *Hash* and Extendable-Output Functions. [Online]. Available: http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf

[7] R.L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, pp. 120-126, 1978

[8] D. Boneh, "Twenty years of attacks on the RSA cryptosystem", *Notices of the American Mathematical Society (AMS)*, vol. 46, no. 2, pp. 203-213, 1999.

[9] A. Nitaj. The Mathematical Cryptography of the RSA Cryptosystem. [Online]. Available: http://www.math.unicaen.fr/~nitaj/RSAnitaj1.pdf

[10] A. I. Ali, "Comparison and evaluation of digital signature schemes employed in NDN network", *International Journal of Embedded systems and Applications(IJESA)*, vol. 5, no. 2, pp. 15-29, Jun. 2015

[11] W. Diffie and M. E. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644-654, Nov. 1976.

[12] A.M. Jaafar, A. Samsudin, "Visual digital signature scheme: a new approach", *IAENG International Journal of Computer Sciences*, vol. 37, no. 4, 2010.

[13] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. (Jan. 2011) Cryptographic sponge functions. [Online]. Available: http://sponge.noekeon.org/CSF-0.1.pdf

[14] E. Milanov. (Jun. 2009) The RSA algorithm. [Online]. Available: https://www.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf.