Laporan Pemodelan
Cryptocurrency

Diajukan sebagai salah satu tugas mata kuliah Pemodelan Matematika
semester II Tahun Akademik 2019-2020

Oleh:

Julius Ferdi	**10117026**
Ryan Nathanael Wongso	**10117058**
Sebastian Adrian Halim	**10117088**
Muhammad Alif Aqsha	**10117108**

Dosen Pembimbing:

Fajar Yuliawan, S.T., M.Si.

PROGRAM STUDI MATEMATIKA

INSTITUT TEKNOLOGI BANDUNG

2020

ABSTRAK

Tidak dapat dipungkiri bahwa uang adalah bagian dari kehidupan manusia yang digunakan untuk melakukan kegiatan transaksi. Selain dari mata uang Fiat, terdapat mata uang kripto yang merupakan mata uang digital yang menggunakan kriptografi untuk menghasilkan uang dan memverifikasi transaksi. Salah satu mata uang kripto yang populer adalah Bitcoin. Pemodelan ini bertujuan untuk memodelkan secara matematis mengenai sistem transaksi, penyimpanan dan keamanan Bitcoin dengan menggunakan pemrograman Python. Pemodelan ini menggunakan studi literatur sebagai landasan utama.

Pada sistem mata uang kripto, pelaku transaksi terbagi menjadi *traders*, yaitu pihak yang melakukan transaksi, dan *miners*, yaitu pihak yang memverifikasi transaksi. Bagian utama keamanan dari *cryptocurrency* ini adalah kunci privat, kunci publik, tanda tangan digital, fungsi *hash*, serta *blockchain*. Manipulasi transaksi pada *cryptocurrency* khususnya Bitcoin sangat sulit dilakukan karena fungsi *hash* yang acak, tanda tangan yang bersifat unik, verifikasi *proof of work* yang harus memenuhi *threshold*, serta pemilihan blok yang valid sebelum dimasukkan ke dalam *blockchain*. Segala sistem keamanan ini kemudian disimulasikan menggunakan pemrograman Python, sehingga terlihat jelas kerumitan dari tiap proses yang menjamin keamanan transaksi.

Kata kunci : *cryptocurrency*, kunci privat, kunci publik, tanda tangan digital, fungsi *hash*, *blockchain*, *proof of work.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I PENDAHULUAN

1.1 Latar Belakang

Dewasa ini, transaksi merupakan salah satu elemen penting dalam kehidupan manusia. Pada umumnya, transaksi-transaksi yang dilakukan sehari-hari menggunakan sistem pembayaran dengan uang. Uang adalah benda-benda yang disetujui oleh masyarakat untuk mengadakan tukar menukar/perdagangan (Sukirno 2016). Salah satu bentuk uang adalah uang fiat. Uang fiat didefinisikan sebagai bentuk uang yang dikeluarkan pemerintah yang memiliki nilai sesuai dengan nilai nominal pada benda tersebut (Rollins 2003), contohnya uang giral dan kartal. Uang fiat memiliki banyak kelebihan, beberapa di antaranya adalah:

1. memudahkan kegiatan tukar menukar: setiap orang dapat bertransaksi tanpa harus menyepakati cara pembayaran tertentu,
2. sebagai satuan nilai: uang dapat dijadikan patokan untuk menentukan besarnya nilai suatu barang dan kualitas barang tersebut, dan
3. sebagai alat penyimpanan nilai: penggunaan mata uang fiat memungkinkan seseorang menyimpan kekayaannya ke dalam bentuk yang lebih praktis (Sukirno 2016).

Terlepas dari segala kelebihannya sebagai alat tukar modern yang digunakan pada masa kini, uang fiat memiliki dua kekurangan utama yaitu: berlaku secara lokal, beberapa uang fiat tidak dapat selalu digunakan secara global dan; regulasinya di atur oleh pihak ketiga seperti pemerintah, bank, dan perusahaan penyedia layanan uang digital.

Dua kekurangan utama dari uang yang selama ini dikenal membuat banyak orang kemudian mulai mencoba menciptakan suatu sistem mata uang baru yang bebas dari sistem sentralisasi dan berlaku secara global. Dalam prosesnya, seorang bernama Satoshi Nakamoto berhasil menciptakan jenis mata uang baru yang dikenal dengan nama Bitcoin. Bitcoin termasuk dalam jenis mata uang kripto, yaitu mata uang digital yang menggunakan kriptografi untuk menghasilkan uang dan memverifikasi transaksi. Sistem desentralisasi yang diusung Bitcoin membuat mata uang ini sangat revolusioner, karena di dalam sistem ini, transaksi tidak di atur oleh pihak ketiga sebagai regulator. Bitcoin juga berlaku secara global dan hingga saat ini relatif aman.

Maka dari itu, dalam pemodelan ini kami ingin mengkaji lebih jauh tentang mata uang kripto. Dalam hal ini, mata uang kripto yang kami maksud adalah Bitcoin. Fokus utama kami adalah menjelaskan dengan cukup detail mengenai sistem transaksi Bitcoin dan sistem keamanan Bitcoin. Hasil pemodelan Bitcoin yang kami lakukan kebanyakan dinyatakan dalam narasi. Walaupun demikian terdapat beberapa notasi dan penjelasan secara matematis.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, dua masalah utama yang ingin kami modelkan adalah:

1. Bagaimana model matematika dari sistem transaksi dan penyimpanan mata uang kripto?
2. Bagaimana menguji sistem keamanan mata uang kripto menggunakan model yang dibangun?

1.3 Tujuan Pemodelan

Tujuan yang ingin dicapai dari pemodelan ini adalah:

1. Memodelkan secara matematis sistem transaksi dan penyimpanan mata uang kripto
2. Menguji keamanan sistem mata uang kripto dengan model yang dibangun.

1.4 Manfaat Pemodelan

Manfaat yang kami harapkan dari pemodelan ini adalah:

1. Sebagai bahan ilmu untuk menambah pemahaman mengenai Bitcoin, khususnya bagi mahasiswa matematik, baik dalam cara kerja maupun konsep matematika dibaliknya dan;
2. Dapat dijadikan acuan untuk penelitian selanjutnya mengenai Bitcoin.

1.5 Batasan Masalah dan Asumsi Pemodelan

Batasan masalah dan asumsi yang ditetapkan pada pemodelan ini adalah:

1. Pemodelan ini hanya akan membahas penyimpanan, transaksi, dan keamanan sistem mata uang kripto
2. Mata uang kripto yang dimodelkan adalah Bitcoin
3. Mayoritas pihak asumsikan sebagai "*good guys*" yaitu pihak yang berusaha meraih keuntungan semaksimal mungkin tapi masih patuh terhadap peraturan yang disepakati
4. Tidak ada hambatan oleh *hardware* dalam proses verifikasi transaksi.

BAB II TEORI DASAR

2.1 Sistem Kripto Asimetrik - Kriptografi Kunci Publik

Sistem kripto asimetrik merupakan suatu skema pengiriman pesan secara aman yang menggunakan kunci publik dan kunci privat (Hoffstein, Pipher dan Silverman 2014). Kedua kunci ini secara fisik merupakan serangkaian angka yang cukup panjang. Perbedaan utamanya terletak pada siapa yang memiliki akses kepada kunci tersebut. Kunci publik, sesuai namanya, dapat diperoleh oleh semua orang, sedangkan hanya pemiliknya sendiri yang memiliki akses kepada kunci privat. Dalam laporan ini, masalah pengiriman pesan yang dibahas adalah bagaimana penerima suatu pesan mampu mengonfirmasi identitas sang pengirim pesan menggunakan kunci publik.

Pada sistem kripto asimetrik, kunci publik diperoleh dari kunci privat. Sistem dirancang sedemikian rupa sehingga mudah untuk memperoleh kunci publik dari kunci privat, namun sangat sulit untuk melakukan sebaliknya. Kunci publik kemudian disebarkan ke seluruh orang (atau jaringan), dan kunci privat dirahasiakan. Kunci publik dapat juga dikatakan sebagai alamat dari pemegang kunci privat yang bersesuaian.

Kunci privat ini berguna untuk “menandatangani” suatu pesan, sehingga ketika pesan dan “tanda tangan” tersebut disebarkan pada jaringan, semua pihak mampu mengonfirmasi, dengan menggunakan kunci publik, bahwa identitas dari pengirim pesan tersebut memang benar seperti yang ia klaim. Dengan kata lain, pihak pada jaringan tersebut mampu mengetahui apakah pengirim pesan tersebut memiliki kunci privat yang cocok tanpa harus mengetahui kunci privat itu sendiri (*zero knowledge proof*).

Sangat sulit bagi pihak *adversary* (orang yang berniat jahat) untuk berpura-pura menjadi orang lain karena supaya “tanda tangan”-nya cocok dan bisa dikonfirmasi jaringan, ia harus memperoleh kunci privat orang lain tersebut yang sangat sulit untuk dilakukan.

2.2 Digital Signature Algorithm (DSA)

Digital Signature Algorithm (DSA) merupakan salah satu sistem kripto asimetrik yang digunakan untuk mengonfirmasi identitas pengirim pesan.

Pada sistem ini, terdapat tiga parameter publik, yaitu bilangan prima p dan q sehingga $p \equiv 1 \bmod q$, dan bilangan bulat g dengan $1 \leq g \leq p-1$ sehingga g memiliki orde q pada modulo p (yaitu q merupakan bilangan bulat terkecil yang memenuhi $1 \leq q \leq p-1$ dan $g^q \equiv 1 \bmod p$).

Kemudian, dilakukan langkah-langkah seperti berikut (Hoffstein, Pipher dan Silverman 2014).

Tabel 1: Skema DSA

Pengirim Pesan	Pelaku Verifikasi
Pembuatan Kunci	
Pilih kunci privat a bilangan bulat dengan $1 \leq a \leq q-1$. Hitung $A = g^a \bmod p$. A ini merupakan kunci publik yang akan disebarkan.	
Tanda Tangan	
Pilih pesan D yang telah diubah menjadi bilangan bulat dan berada pada modulo q. Pilih bilangan bulat k secara acak dengan $1 < k < q$. Hitung tanda tangan $S_1 = (g^k \bmod p) \bmod q$ dan $S_2 = (D + aS_1)k^{-1} \bmod q$, dengan k^{-1} adalah bilangan bulat di antara 1 dan q yang memenuhi $kk^{-1} \equiv 1 \bmod q$.	
Verifikasi	
	Hitung $V_1 = DS_2^{-1} \bmod \text{q}$ dan $V_2 = S_1S_2^{-1} \bmod q$, dengan S_2^{-1} merupakan bilangan bulat positif yang tidak melebihi q dan memenuhi $S_2S_2^{-1} \equiv 1 \bmod q$.

	Periksa apakah $(g^{V_1}A^{V_2} \bmod p) \bmod q = S_1$. Jika benar, tanda tangan diverifikasi. Jika tidak, tanda tangan tidak diverifikasi.

Bukti bahwa algoritma di atas konsisten:

Misalkan pengirim pesan telah menerbitkan dokumen D, tandatangan S_1 dan S_2. Perhatikan bahwa karena $S_2 \equiv (D + aS_1)k^{-1} \pmod q$, maka $S_2^{-1} \equiv k(D + aS_1)^{-1} \pmod q$. Karena g merupakan elemen $\mathbb{Z}p^*$ yang memiliki orde q, maka $g^q \equiv 1 \pmod p$.

Karena $V_1 \equiv DS_2^{-1} \pmod q$, maka $V_1 = dq + DS_2^{-1}$ untuk suatu d bilangan bulat, sehingga $g^{V_1} = g^{dq+DS_2^{-1}} = (g^q)^d g^{DS_2^{-1}} \equiv (1)^d g^{DS_2^{-1}} \equiv g^{DS_2^{-1}} \pmod p$, sehingga $g^{V_1} \equiv g^{DS_2^{-1}} \equiv g^{Dk(D+aS_1)^{-1}} \pmod p$.

Dengan argumen yang serupa, karena $V_2 \equiv S_1 S_2^{-1} \pmod q$ dan $A \equiv g^a \pmod p$, maka $A^{V_2} \equiv (g^a)^{V_2} \equiv g^{aV_2} \equiv g^{aS_1S_2^{-1}} \equiv g^{aS_1k(D+aS_1)^{-1}} \pmod p$. Maka

$$g^{V_1}A^{V_2} \equiv g^{Dk(D+aS_1)^{-1}} g^{aS_1k(D+aS_1)^{-1}} \equiv g^{(D+aS_1)k(D+aS_1)^{-1}} \equiv g^k \pmod p$$

Akibatnya $g^{V_1}A^{V_2} \pmod q \equiv (g^k \pmod p) \pmod q = S_1$, sehingga transaksi akan diverifikasi.

2.3 Fungsi *Hash*

Fungsi *hash* merupakan fungsi yang memetakan dokumen dengan ukuran sembarang ke suatu *string* atau bilangan dengan panjang yang tetap, sehingga fungsi tersebut mudah untuk dihitung, namun sangat sulit untuk dibalikkan (Hoffstein, Pipher dan Silverman 2014).

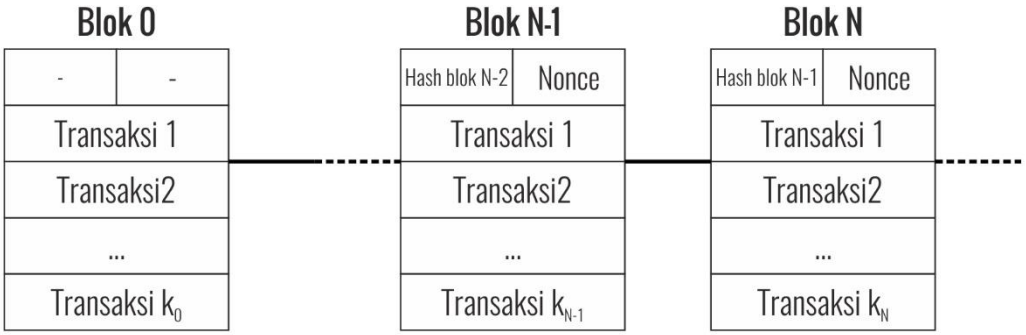
Fungsi *hash* yang digunakan pada kriptografi memiliki karakteristik sebagai berikut:

1. Deterministik, yaitu *input* yang sama akan selalu menghasilkan *output* yang sama.
2. Membutuhkan waktu yang singkat untuk menghitung *output*-nya.
3. Sangat sulit untuk memperoleh *input* dari *output*.
4. Sangat sulit untuk mencari dua *input* berbeda yang menghasilkan *output* identik.
5. Sedikit perubahan saja pada *input* menghasilkan *output* yang jauh berbeda sehingga *output* yang baru terlihat tidak berkorelasi dengan *output* yang lama (*avalanche effect*) (Saif Al-Kuwari 2011).

Salah satu fungsi *hash* yang cukup terkenal adalah SHA-256. Pada pemodelan ini, fungsi *hash* yang akan digunakan adalah SHA-256.

2.4 *Blockchain*

Blockchain adalah sebuah *Ledger* atau diibaratkan dengan sebuah buku besar yang berfungsi sebagai sistem pencatatan transaksi di banyak *database* yang tersebar luas di banyak komputer yang masing-masing memuat catatan yang identik. *Blockchain* bersifat terbuka, aman, dan mudah diakses semua orang. Dengan catatan transaksi yang ter-desentralisasi ini, maka hampir tidak mungkin untuk diretas atau diubah secara sepihak, tanpa menguasai jumlah mayoritas dari semua *database* atau komputer tersebut.



Gambar 1: Ilustrasi *Blockchain*

Catatan transaksi-transaksi ini dimuat dalam blok-blok yang saling tersambung. Jika satu blok sudah penuh, maka akan diciptakan blok berikutnya yang terkoneksi dengan blok sebelumnya. Catatan transaksi yang dimuat di blok yang sudah diciptakan, tidak akan bisa diubah lagi - sehingga *blockchain* sering disebut mempunyai sifat *immutable*. Transaksi yang dicatat memuat tiga komponen utama yaitu alamat Bitcoin pihak yang membayar, termasuk sumber dana Bitcoin yang digunakan, alamat Bitcoin pihak yang menerima dan jumlah Bitcoin yang di transfer.

2.5 *Miners* dan Proses *Mining*

Miners adalah orang atau kelompok yang berusaha menambahkan blok pada *blockchain*. *Miners* akan "mendengarkan/melihat transaksi" yang terjadi pada setiap orang. Setiap transaksi

yang valid akan dipilih *miners* dan ditambahkan ke dalam *blockchain* apabila *miner* sudah melakukan *proof of work*. Pada setiap penambahan blok baru, hanya satu *miner* yang blok versinya ditambahkan ke *blockchain*.

Pemilihan blok yang akan ditambahkan sedikit mirip dengan sistem *lottery*. *Miner* yang terpilih adalah yang pertama kali menyelesaikan *proof of work*, yaitu usaha membangkitkan suatu *hash output* (basis 16) dengan fungsi SHA-256 di bawah *threshold* tertentu. Pada setiap blok (kecuali blok 0 atau genesis) terdapat suatu bilangan yang disebut dengan *nonce.* Setiap *miner* menambah *nonce* dengan jumlah tertentu sehingga *output* dari *hash* blok yang dihasilkan (termasuk *nonce*-nya) di bawah *threshold*. *Hash output* tersebut dapat dikatakan sebagai "solusi".

Blok yang dimasukkan ke dalam *blockchain* adalah blok dengan *hash* di bawah *threshold* tercepat yang ditemukan *miner* yang telah melakukan *proof of work*. Jika ia berhasil, maka *miner*s lainnya akan menerima blok *miner* tersebut setelah verifikasi transaksi *payers* dan jumlah Bitcoin yang dikreditkan ke *miner* tersebut (jika ada transaksi *payers* yang salah dicatat atau jumlah Bitcoin yang di-"tambang" oleh *miners* tersebut lebih dari 25, *miners* lainnya akan menolak blok tersebut).

Jika penemuan "solusi" cenderung membutuhkan waktu kurang dari 10 menit, sistem akan menurunkan *threshold* (sehingga peluang mendapatkan "solusi" menjadi lebih kecil), dan juga sebaliknya, jika cenderung lebih dari 10 menit, sistem akan menaikkan *threshold* (sehingga peluang mendapatkan "solusi" menjadi lebih besar). *Miner* yang blok versinya ditambahkan akan mendapatkan hadiah berupa Bitcoin secara otomatis dari sistem serta biaya transaksi yang diberikan oleh setiap pelaku transaksi.

BAB III HASIL PEMODELAN

3.1 Pelaku Transaksi

Para pelaku transaksi Bitcoin terbagi menjadi *traders* dan *miner*s. *Traders* adalah semua orang yang melakukan transaksi (mengirim/menerima ke/dari pihak lain) Bitcoin, sedangkan *miner*s adalah sekelompok orang yang melakukan verifikasi transaksi dan berlomba-lomba menyelesaikan tugas tertentu dan akan mendapatkan imbalan berupa sejumlah Bitcoin.

Pada pemodelan ini, misalkan pelaku transaksi terdiri dari A, B, C, D, dan E. A dan B adalah orang yang melakukan transaksi dengan saldo masing-masing adalah sebanyak nol dan lima puluh koin, sedangkan C, D dan E adalah *miners* dengan kekuatan komputer yang tidak berbeda secara signifikan.

3.2 Pemilihan Parameter Publik untuk DSA

Salah satu cara untuk membangkitkan parameter publik yang sesuai untuk DSA adalah dengan menggunakan pemrograman Python sebagai berikut.

```
# Algoritma penentuan parameter publik p, q, dan q

def GeneratePrimePair():
    """ Fungsi yang membangkitkan bilangan prima p dan q
    dengan p = 2q + 1 sehingga p = 1 mod q
    """
    ispairprime = False
    while ispairprime == False:
        q = random.randint(2**128, 2**129)
        if sympy.isprime(q) == True:
            if sympy.isprime(2*q+1) == True:
                p = 2*q+1
                ispairprime = True
    return p, q

def GenerateElement(p, q):
    """Meng-generate bilangan g yang memiliki orde q pada modulo p
    Perhatikan bahwa p = 2q+1, sehingga untuk setiap t di Z_p*, (t^2)^q = t^(2q) = t^(p-1) = 1 mod p
    Jika t^2 != 0 atau 1 mod p, maka t^2 berorde tepat q, karena jika r orde dari t^2, haruslah r habis membagi q
    Maka, r = 1 atau q. Tapi, t^2 != 1 mod p, sehingga haruslah r = p
    """
    t = random.randint(1, p-1)
    # pow(a,e,b) menghasilkan a^e modulo b
    # fungsi pow(a,e,b) digunakan karena jauh lebih efisien dibandingkan memangkatkan a^e, kemudian dimodulo b-kan.
    g = pow(t,2,p)
    while g == 1:
        t = random.randint(1, p-1)
        g = pow(t,2,p)
    return g
```

Gambar 2: Pemilihan Parameter Publik(a)

```
p, q = GeneratePrimePair()
print(p, q)
```

```
111667424642583579118676348409985127036 3 558337123212917895593381742049925635181
```

```
g = GenerateElement(p, q)
print(g)
```

```
606189179136508625129240953541620883235
```

Gambar 3: Pemilihan Parameter Publik(b)

Diperoleh parameter publik

$p = 111667424642583579118676348409985127036 3,$

$q = 558337123212917895593381742049925635181$

$g = 606189179136508625129240953541620883235.$

Bukti bahwa g berorde q modulo p:

Dari algoritma di atas, dapat dilihat bahwa $g \equiv t^2 \bmod p$, dengan $g \not\equiv 1 \bmod p$. Perhatikan bahwa t bilangan bulat positif dengan $1 \leq t \leq p-1$, sehingga t saling prima dengan p. Maka, dengan Teorema Euler, $t^{p-1} \equiv 1 \bmod p$. Tetapi, $p = 2q + 1$, sehingga $t^{2q} \equiv 1 \bmod p \Rightarrow g^q \equiv 1 \bmod p$.

Misalkan s adalah orde dari g. Maka s adalah bilangan bulat terkecil yang memenuhi $1 \leq s \leq p-1$ dan $g^s \equiv 1 \bmod p$. Perhatikan bahwa q dan s bilangan bulat positif, sehingga dengan pembagian Euclidean, terdapat bilangan bulat non-negatif d dan r dengan $0 \leq r \leq s-1$ sehingga $q = ds + r$. Akibatnya, $1 \equiv g^q \equiv g^{ds+r} \equiv (g^s)^d g^r \equiv (1)^d g^r \equiv g^r \bmod p$. Jika $r \neq 0$, maka terdapat bilangan bulat positif r dengan $1 \leq r \leq s-1 < s \leq p-1$ sehingga $g^r \equiv 1 \bmod p$, kontradiksi dengan asumsi bahwa s adalah bilangan bulat terkecil yang memenuhi $1 \leq s \leq p-1$ dan $g^s \equiv 1 \bmod p$. Maka haruslah $r = 0$, sehingga s membagi habis q.

Karena q prima, maka $s = 1$ atau $s = q$. Tetapi, $g^1 \equiv t^2 \not\equiv 1 \bmod p$, sehingga haruslah $s = q$. Terbukti bahwa q orde dari g.

3.3 Pembuatan Akun

Langkah awal dalam pembuatan akun Bitcoin yang sesungguhnya adalah memilih media/aplikasi untuk mengelola Bitcoin (*wallet*). Untuk *hardware* dapat berupa Ledger Nano X, Trezor T, dan Keepkey sedangkan untuk *software* dapat berupa Bitcoin Core, Electrum, dan Coinbase. Kemudian, hal selanjutnya adalah melakukan *sign up* dan melakukan *ID Verification* untuk beberapa *software* (beberapa *software* tidak melakukan hal ini atau secara anonim saja). Selanjutnya, orang tersebut akan mendapatkan *address* Bitcoin. Proses pembuatan akun telah selesai dan dapat digunakan untuk melakukan transaksi.

Untuk kepentingan pemodelan ini, akan digunakan pembuatan *address* (untuk penyederhanaan dianggap sama dengan kunci publik) dan kunci privat yang lebih sederhana dengan memanfaatkan DSA. Berikut adalah implementasinya pada Python beserta keluarannya.

```
# Membuat address (Kuncil publik) beserta kunci privatnya

from random import randint

p = 111667424642583579118676348409985127036 3
q = 558337123212917895593381742049925635181
g = 606189179136508625129240953541620883235

a = randint(1, q-1)
A = pow(g,a,p)
print("Your address: ", A, "\nYour private key: ", a, "\nDo not share your private key to anyone!")
```

```
Your address:  101899096962698437412079556025070525898 3
Your private key:  237043361745808634896709823243487876099
Do not share your private key to anyone!
```

Gambar 4 : Konstruksi kunci publik dan kunci privat

Setelah algoritma di atas diulang beberapa kali, diperoleh kunci publik/alamat dan kunci privat setiap pelaku transaksi pada pemodelan ini.

Tabel 2: Kunci Publik dan Kunci Privat Pemain Bitcoin

Pemain	Kunci Publik/Alamat	Kunci Privat
A	498063637122669310186914979135247239947	361673680473608126109674134178452992434
B	793833148500065718802366039211618163182	494958898825128152883083981490204250529
C	105004471294980766237195361488000655536 3	369496142818372634830485004976076376795
D	101899096962698437412079556025070525898 3	237043361745808634896709823243487876099
E	247686779170034366557143912312509136161	155502889988157414713048991567409255940

3.4 Penyimpanan Mata Uang Kripto

Bitcoin tidak menyimpan uangnya dalam bentuk akun dengan informasi saldonya, melainkan dalam bentuk UTXO (*unspent transaction outputs*) (Monetha 2019). Suatu transaksi Bitcoin memiliki *input* dan *output*. *Input* merupakan informasi *output* yang belum dibelanjakan (*unspent*) dan hendak digunakan untuk melakukan transaksi, dan *output* merupakan informasi alamat tujuan transaksi dan jumlahnya.

Sebagai ilustrasi yang sesuai dengan kondisi awal kita, misalkan B telah menerima transfer sebesar 50 koin, dan ia belum membelanjakan 50 koin tersebut. Maka transaksi yang B terima tersebut disebut UTXO dan hanya bisa dibelanjakan oleh B sendiri.

Misalkan B ingin mengirimkan 35 koin kepada A. Untuk melakukan pengiriman koin tersebut, B memasukkan informasi mengenai 50 koin UTXO tadi sebagai *input* (misalkan *hash* dan indeks dari UTXO tersebut).

Tidak seperti pada model akun dan saldo yang umumnya digunakan pada bank konvensional, B tidak bisa mengambil 35 koin dari 50 koin pada UTXO tersebut untuk dikirimkan ke A. B harus menggunakan seluruh 50 koin pada UTXO tersebut. Maka, untuk mengakalinya, B mengirimkan seluruh 50 koin tersebut dengan rincian 35 koin kepada A, dan 15 koin sisanya ke dirinya sendiri kembali. Semua informasi mengenai alamat tujuan dan jumlah yang dikirimkan ini dimasukkan sebagai *output*.

Ketika transaksi ini berhasil, maka UTXO berisikan 50 koin tadi akan menjadi *spent transaction* dan tidak dapat dibelanjakan kembali, sedangkan *output* pada transaksi yang baru saja dilakukan B akan menjadi UTXO yang baru, dengan rincian sebuah UTXO berisikan 35 koin yang hanya bisa dibelanjakan oleh A dan sebuah UTXO berisikan 15 koin yang hanya bisa dibelanjakan oleh B.

3.5 Proses Transaksi

B memiliki UTXO sebesar 50 koin, dan ia hendak mengirimkan 35 koin kepada A. Maka B perlu menyiapkan *hash* dan indeks dari UTXO tersebut sebagai *input*, kunci privat miliknya yang

diasosiasikan dengan UTXO tersebut, beserta *output* yang menyatakan bahwa ia hendak mengirimkan 35 koin kepada A dan 15 sisanya ke dirinya sendiri. Rincian transaksinya adalah:

Tabel 3: Rincian Transaksi

Unspent Transaction:	#*hash* UTXO yang ingin digunakan
Index:	#indeks dari UTXO di atas yang ingin digunakan
Transfer to:	[[A, 35 koin], [B, 15 koin]]
Hash:	#*hash* dari gabungan ketiga informasi di atas
Signature:	[S1, S2] #tanda tangan yang diperoleh dengan DSA

Berikut ini adalah implementasinya pada pemrograman Python.

```
from hashlib import sha256
from random import randint
```

```
# Parameter publik pada DSA
p = 11166742464258357911867634840998512703 63
q = 55833712321291789559338174204992563518 1
g = 60618917913650862512924095354162088323 5
```

```
class Transaction:
    def __init__(self, inp, inpidx, inpkey, outp):
        self.inp = inp # hash dari unspent transaction yang ingin dibelanjakan
        self.inpidx = inpidx # indeks dari unspent transaction di atas yang ingin dibelanjakan
        self.inpkey = inpkey # kunci privat yang diasosiasikan dengan pemilik unspent transaction dengan indeks di atas
        self.outp = outp # penerima yang dituju & jumlah koin yang ingin ditransfer
        self.toBeHashed = str(self.inp) + str(self.inpidx) + str(self.outp) #informasi transaksi yang akan di-hash
        self.hash = sha256(self.toBeHashed.encode('utf8')).hexdigest()
        global p
        global q
        global g
        # Tanda tangan digital dengan DSA
        D = int(self.hash, 16)
        k = randint(2, q-1)
        S1 = pow(g,k,p)%q
        S2 = (D+self.inpkey*S1)*(pow(k,q-2,q))%q
        self.sign = [S1, S2]
    def __str__(self):
        string = "Unspent Transaction: " + str(self.inp) + "\n"
        string += "Index: " + str(self.inpidx) + "\n"
        string += "Transfer to: " + str(self.outp) + "\n"
        string += "Hash: " + str(self.hash) + "\n"
        string += "Signature: " + str(self.sign)
        return string
```

Gambar 5: Implementasi Transaksi(a)

```
inp = '1a428822908b32a273d294ad9a2a30e79b4a812d5f79cb98b73f4c406a1a3643' #hash transaksi yang mau dipakai
inpidx = 0 #index dari transaksi input yang ingin dipakai
inpkey = 4949588988251281528830839814902042505 29  #kunci privat yang diasosiasikan dengan address pelaku transaksi di atas
O1 = 498063637122669310186914979135247239947 #alamat A
O2 = 793833148500065718802366039211618163182 #alamat B
outp = [[O1, 35], [O2, 15]] #address yang dituju beserta jumlah yang ingin ditransfer untuk masing-masing address
transaction = Transaction(inp, inpidx, inpkey, outp)
```

```
print(transaction)
```

```
Unspent Transaction: 1a428822908b32a273d294ad9a2a30e79b4a812d5f79cb98b73f4c406a1a3643
Index: 0
Transfer to: [[498063637122669310186914979135247239947, 35], [793833148500065718802366039211618163182, 15]]
Hash: 3114d9838781b5d2ef9b9abaee6bc7e1531345098fcf109d7968e8710392c377
Signature: [55728162667160948598341541448702799816 3, 201672271114419374600637216245797012022]
```

```
import openpyxl
```

```
wb = openpyxl.load_workbook('awaiting.xlsx')
ws = wb.worksheets[0]
rows = [[str(transaction.inp),
       str(transaction.inpidx),
       str(transaction.outp),
       str(transaction.hash),
       str(transaction.sign)]]
for row in rows:
    ws.append(row)
wb.save('awaiting.xlsx')
```

Gambar 6: Implementasi Transaksi(b)

Informasi transaksi ini dimasukkan ke suatu *transaction pool* berbentuk *file* “awaiting.xlsx” yang berisikan semua transaksi yang akan diverifikasi oleh *miner*.

3.6 Verifikasi

Miner akan memverifikasi seluruh transaksi pada "awaiting.xlsx" dengan algoritma verifikasi tanda tangan yang telah ditetapkan di awal. Pada pemodelan ini, karena tanda tangan setiap transaksi dihasilkan dari DSA, maka algoritma yang digunakan untuk verifikasi juga dari DSA. Berikut adalah implementasinya pada pemrograman Python.

```
from hashlib import sha256
import openpyxl
import xlrd
from ast import literal_eval
```

```
# Parameter publik untuk DSA
p = 1116674246425835791186763484099851270363
q = 55833712321291789559338174204992563518l
g = 60618917913650862512924095354162088323 5
```

```
def Verify(outphash,A,sign):
    """ Verifikasi transaksi dalam bentuk outphash dengan address pengirim adalah A dan tandatangan sign = S1, S2
    """
    global p
    global q
    global g
    D = int(outphash, 16)
    S1 = sign[0]
    S2 = sign[1]
    V1 = (D*pow(S2,q-2,q))%q #S2^(q-2) = S2^(-1) mod q
    V2 = (S1*pow(S2,q-2,q))%q
    return ((pow(g,V1,p)*pow(A,V2,p))%p)%q == S1 #signature diverifikasi benar jikka persamaan di samping bernilai benar
```

```
def is_inp_there(path, inp):
    """ Mencari apakah suatu hash input terdapat pada daftar unspent transactions
    dan memberikan nomor baris beserta data pada baris tersebut
    """
    wb = xlrd.open_workbook(path)
    sheet = wb.sheet_by_index(0)

    isThere = False
    for row_num in range(sheet.nrows):
        row_data = sheet.row_values(row_num)
        if row_data[0] == inp:
            isThere = True
            break
    return isThere, row_num, row_data
```

Gambar 7: Verifikasi Transaksi(a)

```
wb = xlrd.open_workbook("awaiting.xlsx")
sheet = wb.sheet_by_index(0)

for row_number in range(sheet.nrows):
    inp = sheet.row_values(row_number)[0]
    inpidx = int(sheet.row_values(row_number)[1])
    outp = literal_eval(sheet.row_values(row_number)[2])
    sign = literal_eval(sheet.row_values(row_number)[4])
    hash0 = sha256((str(inp)+str(inpidx)+str(outp)).encode('utf8')).hexdigest()
    print("\n" + "Unspent hash to be transferred: " + inp + "\n"
          + "Index: " + str(inpidx) +"\n"
          + "Transfer to: " + str(outp) + "\n"
          + "Signature: " + str(sign) + "\n"
          + "Hash: " + hash0)
    transaction_sum = 0
    for i in range (0,len(outp)):
        transaction_sum += outp[i][1]
    isThere, row_num, row_data = is_inp_there("unspent.xlsx", inp)
    unspent_owners = literal_eval(row_data[1])
    isUnspent = isThere & (inpidx >= 0) & (inpidx < len(unspent_owners)) & (unspent_owners[inpidx] != 0)
    if isUnspent == False:
        print("Input is not verified \n")
    else:
        sender_address = unspent_owners[inpidx][0]
        if transaction_sum > unspent_owners[inpidx][1]:
            print("Transaction amount exceed unspent amount \n")
        else:
            if Verify(hash0,sender_address,sign) == False:
                print("Signature is not verified \n")
```

```
            else:
                print('Transaction is verified \n')
                #Menghapus transaksi yang dipakai untuk input dari unspent output
                unspent_owners[inpidx] = 0
                wb = openpyxl.load_workbook('unspent.xlsx')
                ws = wb.worksheets[0]
                ws.cell(row=row_num+1, column=2).value = str(unspent_owners)
                #Menambahkan transaksi baru ke unspent output
                rows = [[hash0, str(outp)]]
                for row in rows:
                    ws.append(row)
                wb.save('unspent.xlsx')
                wb = openpyxl.load_workbook('newblock.xlsx')
                ws = wb.worksheets[0]
                rows = [[str(sender_address), str(outp), hash0]]
                for row in rows:
                    ws.append(row)
                wb.save('newblock.xlsx')
```

```
Unspent hash to be transferred: 1a428822908b32a273d294ad9a2a30e79b4a812d5f79cb98b73f4c406a1a3643
Index: 0
Transfer to: [[49806363712266931018691497913524723994 7, 35], [79383314850006571880236603921161816318 3182, 15]]
Signature: [55728162667160948598341541448702799816 3, 20167227111441937460063721624579701202 2]
Hash: 3114d9838781b5d2ef9b9abaee6bc7e1531345098fcf109d7968e8710392c377
Transaction is verified
```

```
#Menghapus transaksi dari awaiting list
book = openpyxl.Workbook()
sheet = book.active

book.save("awaiting.xlsx")
```

Gambar 8: Verifikasi Transaksi(b)

Transaksi yang dimasukkan ke blok baru versi setiap *miner* hanyalah transaksi yang diverifikasi.

3.7 *Proof of Work* & Penyambungan Blok

Setelah beberapa saat, para *miner* akan berlomba-lomba untuk memenangkan blok mereka supaya disambungkan ke *blockchain* utama. Para *miner* ini akan berusaha menemukan suatu *proof of work* yang sangat sulit untuk diperoleh, namun mudah untuk diverifikasi kebenarannya. Seperti yang telah dijelaskan sebelumnya, setiap blok (kecuali pada blok 0 atau blok *genesis*) memiliki suatu bilangan yang disebut *nonce*. Setiap *miner* berlomba-lomba untuk menemukan suatu *nonce* sehingga ketika *nonce* tersebut diletakkan pada blok baru mereka, kemudian blok tersebut di-*hash*, akan dihasilkan suatu *hash* yang berada di bawah batas atas (*threshold*) tertentu. Berikut adalah implementasinya pada pemrograman Python beserta tampilan bloknya, dengan kasus *miner* C memenangkan bloknya.

```
from hashlib import sha256
import openpyxl
import time
```

```
# Menambahkan transaksi mining dari COINBASE ke miner pada blok yang berusaha ditambahkan
wb = openpyxl.load_workbook('newblock.xlsx')
ws = wb.worksheets[0]
inp = "COINBASE"
outp = [[105004471294980766237195361488000655536 3, 25]]
rows = [["COINBASE", str(outp), sha256((str(inp)+str(outp)).encode('utf8')).hexdigest()]]
for row in rows:
    ws.append(row)
wb.save('newblock.xlsx')
```

```
# Threshold integer dari hash
threshold = 2**243
```

```
# Mencari nonce
nonce = 0
isTrue = False
start_time = time.time()
while isTrue == False:
    wb = openpyxl.load_workbook('newblock.xlsx')
    ws = wb.worksheets[0]
    ws['B1'] = nonce
    wb.save('newblock.xlsx')
    with open("newblock.xlsx", "rb") as f:
        bytes = f.read() # read entire file as bytes
        block_hash = sha256(bytes).hexdigest()
    isTrue = int(block_hash, 16) < threshold
    nonce += 1
end_time = time.time()
nonce = nonce - 1
print("Your block wins with nonce = " + str(nonce) + " in " + str((end_time - start_time)/60) + " minutes ")
```

```
Your block wins with nonce = 3976 in 1.306838850180308 minutes
```

```
from os import rename
from os import path
from os import getcwd
```

```
# Mengubah nama file newblock.xlsx menjadi hash block tersebut
oldfile = path.join(getcwd(), "newblock.xlsx")
newfile = path.join(getcwd(), block_hash+".xlsx")
rename(oldfile, newfile)
```

Gambar 9: Implementasi *Proof of Work* dan Penyambungan *Blockchain*(a)

```
import xlrd

wb = xlrd.open_workbook(block_hash + ".xlsx")
sheet = wb.sheet_by_index(0)
printed_block = "\n" + "Block of hash " + block_hash + "\n" + "\n"
printed_block += "Hash of previous block: " + sheet.row_values(0)[0] + "\n"
printed_block += "Nonce: " + str(sheet.row_values(0)[1]) + "\n" + "\n"
for row in range(1, sheet.nrows):
    printed_block += "Transaction " + str(row) + "\n"
    printed_block += "From: " + str(sheet.row_values(row)[0]) + "\n"
    printed_block += "To & Amount: " + str(sheet.row_values(row)[1]) + "\n"
    printed_block += "Hash: " + sheet.row_values(row)[2] + "\n" + "\n"
print(printed_block)
```

```
Block of hash 00022535eba1d810163866c432e05a74e2705cf7fc66a539b51c628787f11437

Hash of previous block: 131dbbce60547c93b8fdb80b973c62f0c471619a44ed46ed711f1470f7c6d0cc
Nonce: 3976.0

Transaction 1
From: 7938331485000657188023660392116181631 82
To & Amount: [[498063637122669310186914979135247239947, 35], [7938331485000657188023660392116181631 82, 15]]
Hash: 3114d9838781b5d2ef9b9abaee6bc7e1531345098fcf109d7968e8710392c377

Transaction 2
From: COINBASE
To & Amount: [[105004471294980766237195361488000655536 3, 25]]
Hash: 1bbdfaaa54418b4a4f68e3287353d9ffee46d12f1af48e019769c1ec440cb156
```

```
# Menambahkan transaksi mining di mana miner menerima 25 koin ke unspent transactions
wb = openpyxl.load_workbook('unspent.xlsx')
ws = wb.worksheets[0]
rows = [[sha256((str(inp)+str(outp)).encode('utf8')).hexdigest(), str(outp)]]
for row in rows:
    ws.append(row)
wb.save('unspent.xlsx')
```

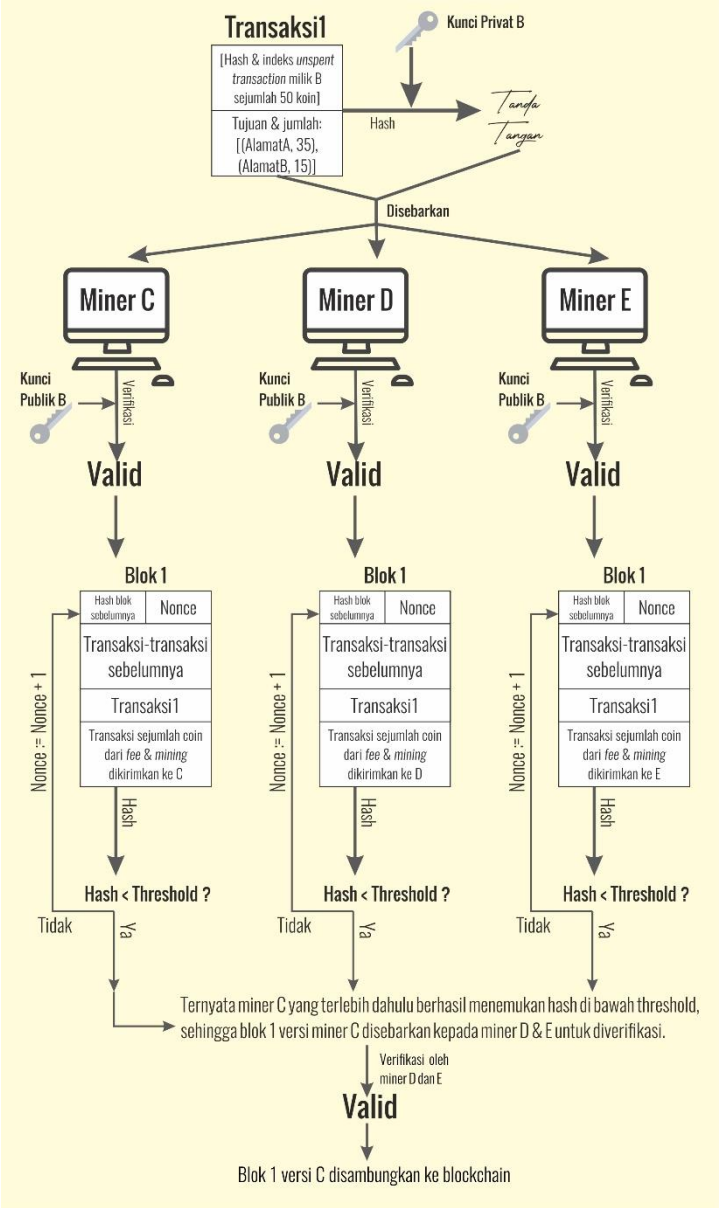
```
# Membuat blok kosong selanjutnya yang berisi hash blok yang berhasil ditambahkan di atas
book = openpyxl.Workbook()
sheet = book.active

sheet['A1'] = block_hash

book.save("newblock.xlsx")
```

Gambar 10: Implementasi *Proof of Work* dan Penyambungan *Blockchain*(a)

Berikut ini adalah diagram yang menggambarkan seluruh proses di atas.



Gambar 11: Skema Transaksi

BAB IV ANALISIS KEAMANAN

4.1 Skenario Serangan terhadap Tanda Tangan Digital

Setelah skenario pada bab III dijalankan, kondisi sekarang adalah B mempunyai 15 koin dan A mempunyai 35 koin. A hendak mengirimkan uang sebesar 10 koin kepada B. Karena satu dan lain hal, *miner* C ingin memanipulasi detail transaksi pada blok yang akan ditambahkan ke *blockchain* sehingga seakan-akan A mengirimkan seluruh 35 koinnya kepada B, bukan 10 (mungkin B adalah teman dekat C ataupun B adalah alamat kedua dari C itu sendiri).

A hendak mengirimkan 10 koin kepada B. Pertama, Ia akan masuk ke dalam *wallet account*-nya dan memilih alamat B sebagai tujuan pengiriman uang. Kemudian, A mengatur jumlah uang yang akan dikirimkan. Pada tahap ini, transaksi A melalui beberapa hal berikut:

Dari belakang layar, sistem akan menggunakan suatu fungsi *hash* (dalam hal ini SHA256) untuk mengubah informasi transaksi (Alamat A, Alamat B, jumlah transaksi). Kemudian, kunci privat dari A digunakan untuk menandatangani transaksi. Tanda tangan ini dilakukan secara digital menggunakan sebuah algoritma (DSA).

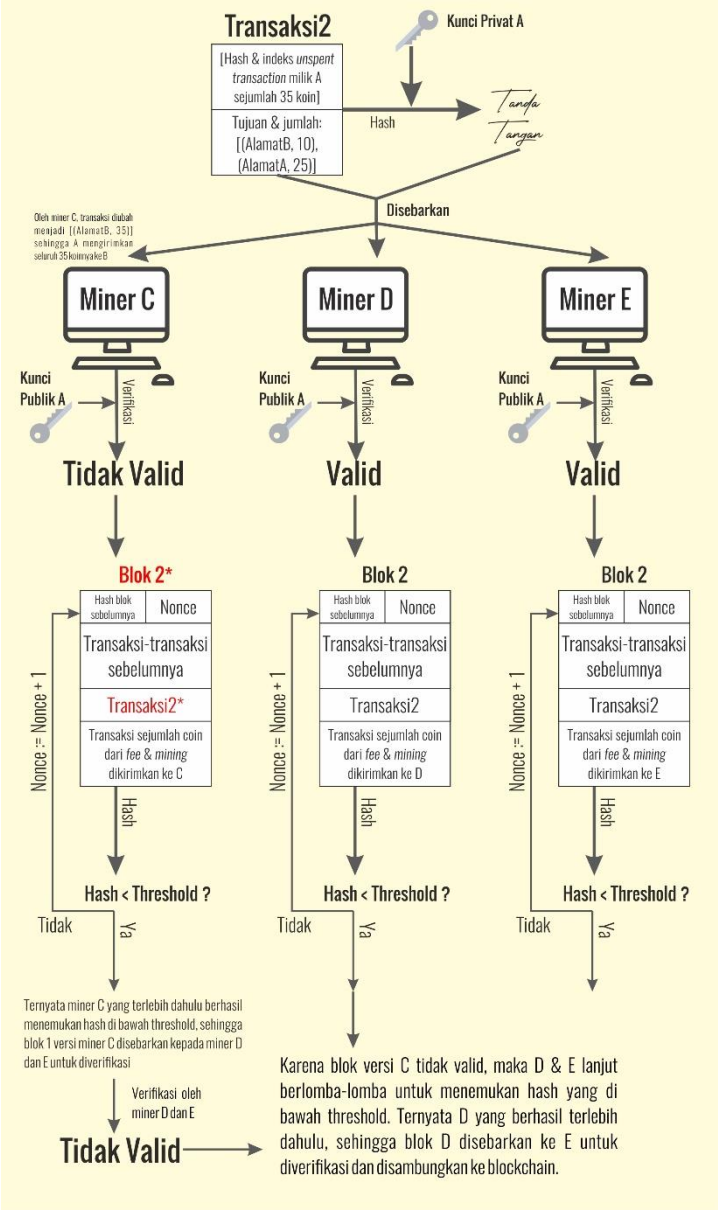
Transaksi yang sudah ditandatangani akan disebarkan ke semua orang (A, B, C, D, E) dan masuk ke sebuah *transaction pool*. Setelah itu, para *miner* (C, D, E) mengecek apakah tanda tangan tersebut valid dengan menggunakan kunci publik dari A. Para *miner* juga mengecek apakah A mempunyai koin yang cukup untuk mengirimkan 10 koin berdasarkan data histori dari *blockchain* sebelumnya.

Saat dicek oleh D dan E, ternyata tanda tangan tersebut valid dan A mempunyai koin yang cukup. Maka D dan E memasukkan transaksi tersebut dalam blok baru masing-masing yang belum disambungkan ke *blockchain* utama. Akan tetapi, *miner* C ingin membantu A berbuat curang, sehingga *miner* C dengan sengaja mengubah detail transaksi seolah-olah A mengirimkan 35 koin kepada B pada blok yang ia susun. Tentu saja, transaksi ini tidak akan bisa divalidasi karena tanda tangan sebenarnya hanya valid untuk 10 koin saja.

Pada saat yang sama C, D, dan E berlomba-lomba untuk mendapatkan *proof of work* yang valid (dibawah *threshold* yang diminta) dan tercepat dari *miner* lainnya. Ternyata secara tidak

disengaja, *miner* C yang tercepat mendapatkan *proof of work* yang sesuai sehingga blok dari *miner* C, yang berisi informasi transaksi dari A yang palsu, ditambahkan ke *blockchain*.

Namun, D dan E yang melihat bahwa ada transaksi yang tidak divalidasi (yaitu transaksi palsu A) tetap berlomba-lomba mencari *proof of work*. Pada akhirnya, *miner* D yang mendapatkan *proof of work* lebih dahulu dari E dan menambahkan blok versi dirinya ke *blockchain* yang tidak berisi transaksi palsu A.



Gambar 12: Skema Skenario Kecurangan

Pada tahap ini, *blockchain* memiliki dua cabang, yaitu blok C dan blok D. Tentu saja, *miner* lainnya, yaitu E dan D sendiri akan melanjutkan blok milik D karena blok tersebut berisi transaksi yang valid. Jika C ingin berhasil berbuat curang, maka ia harus terus menerus menjadi yang tercepat menambahkan blok baru setelah blok dia sendiri dan menjadi *blockchain* yang terpanjang. Namun tentu saja, hal ini sangat sulit dilakukan.

Pada akhirnya *blockchain* yang diusahakan oleh D dan E yang akan lebih panjang dan para *traders* (juga D dan E) akan mempercayai dan memilih *blockchain* tersebut dibandingkan *blockchain* C. Usaha C untuk curang gagal dan D mendapat mendapatkan komisi dari transaksi B dan sejumlah tetap Bitcoin hasil *mining*. Pada tahap ini, B akan menerima notifikasi bahwa A telah mengirimkan 10 koin kepada dirinya, dan A dapat melihat bahwa transaksi tersebut berhasil.

4.2 Analisis Skenario Serangan

Skenario serangan di atas dapat digagalkan karena dua hal, yaitu tanda tangan digital dan *proof of work*. C tidak mampu melakukan kecurangan tersebut karena tanda tangan A hanya valid untuk mengirimkan 10 koin kepada B, bukan 35 koin. Fungsi *hash* pun turut memperkuat keamanan ini karena *hash* yang dihasilkan dari transaksi 10 koin versus 35 koin sangat berbeda, sehingga tanda tangan valid yang dihasilkan pun juga akan sangat berbeda (sangat tidak mungkin untuk ditebak).

Di samping itu, *proof of work* yang dikerjakan merupakan permasalahan yang sangat sulit didapatkan secara komputasional. Agar C mampu melaksanakan serangannya, ia harus selalu menjadi yang tercepat menghasilkan hash di bawah *threshold* pada setiap blok selanjutnya supaya *blockchain* yang memuat blok versinya menjadi versi *blockchain* yang terpanjang. Hal sangat tidak mungkin dilakukan karena untuk melakukan hal ini C harus memiliki kekuatan komputasi jauh lebih besar dibandingkan *miner* lainnya.

4.3 Skenario Serangan terhadap *Blockchain*

Miner C masih belum puas berbuat curang, ia ingin memanipulasi suatu blok yang urutannya jauh lebih dulu dibandingkan blok yang sedang dikerjakan sekarang sehingga berisi

- A mengirim 35 koin ke C
- B mengirim 15 koin ke C

Kedua transaksi ini tidak ditandatangani baik oleh A maupun B sehingga tidak valid. Tentu saja, C kembali berusaha mendapatkan *proof of work* dari transaksi yang tidak valid tersebut. Setelah mendapatkan *proof of work* tersebut, blok versi C ini dimasukkan sebagai pengganti blok tersebut.

4.4 Analisis Skenario Serangan

Skenario serangan di atas dapat digagalkan karena dua hal, yaitu tanda tangan digital dan *proof of work*. C tidak mampu melakukan kecurangan tersebut karena tanda tangan A dan B tidak valid sehingga ketika bloknya ditambahkan, bloknya juga akan menjadi. tidak valid. Fungsi *hash* pun turut memperkuat keamanan ini karena *hash* sehingga tanda tangan valid sangat tidak mungkin untuk ditebak. Selain itu, *hash* yang dihasilkan dari *proof of work* yang dikerjakan C akan berbeda dengan informasi *hash* pada blok selanjutnya. *Hash* dari blok selanjutnya hanya akan cocok dengan *hash* milik blok awal, sebelum diganti dengan blok milik C.

4.5 Analisis Keamanan DSA

Belum ada algoritma yang efisien untuk menyelesaikan permasalahan logaritma diskrit (yaitu mencari kunci privat a yang berkorespondensi dengan kunci publik A sehingga $g^a \equiv A \bmod p$). Hingga saat ini, rekor yang berhasil dipecahkan dalam penyelesaian permasalahan logaritma diskrit adalah pada 12 Desember 2019, di mana sekelompok kriptografer menyelesaikan permasalahan tersebut untuk p berukuran 240 digit (795 bit) dengan algoritma yang membutuhkan 4000 core-years atau 4000 CPU yang dijalankan selama setahun (Goodin 2019).

Namun, serangan di atas berlaku jika *adversary* (penyerang) tidak memiliki akses terhadap histori dokumen dan tanda tangan dari targetnya. Terdapat jenis serangan lainnya terhadap DSA

lainnya yang memanfaatkan histori dokumen dan tanda tangan dari targetnya . Blake pada paper-nya yang berjudul *On the Security of Digital Signature Algorithm* berhasil melakukan serangan *key recovery* dalam waktu kurang dari 5 detik untuk q berukuran 500-bit. Hal ini mengakibatkan DSA tidak aman lagi.

4.6 Analisis Efisiensi *Proof of Work*

Metode *proof of work* yang digunakan pada model ini (dan Bitcoin) dianggap membuang-buang kekuatan komputasi karena menyelesaikan permasalahan yang tidak memiliki kontribusi apapun dalam perkembangan teknologi, khususnya pada kriptografi. Krause pada *paper*-nya yang berjudul *Quantification of energy and carbon costs for mining cryptocurrencies* mengatakan bahwa biaya energi yang diperlukan untuk melakukan *mining* mata uang kripto melebihi biaya energi yang diperlukan untuk menambang logam fisik seperti tambang dan emas.

BAB V PENUTUP

5.1 Simpulan

1) Berdasarkan hasil pemodelan, model dari sistem transaksi dan penyimpanan mata uang kripto yang dibangun adalah:
    a. Masuk ke *wallet* dan melakukan pemilihan jumlah koin serta alamat tujuan
    b. Konversi informasi dengan fungsi *hash*
    c. Tanda tangan digital oleh pengirim dengan kunci privat
    d. Verifikasi tanda tangan oleh *miner* dengan kunci publik pengirim
    e. Verifikasi jumlah koin pengirim oleh *miner*
    f. Melakukan proses *mining*
    g. Penambahan blok versi *miner* tercepat ke *blockchain*
    h. Koin diterima di alamat tujuan
2) Pengujian sistem keamanan mata uang kripto dengan model yang dibangun adalah melalui analisis keamanan DSA dan *proof of* work. Model mata uang kripto yang dibangun belum aman dan efisien.

5.2 Saran

Berdasarkan hasil pemodelan, terdapat beberapa saran yang dapat digunakan untuk mengembangkan model yang sudah dibangun yaitu:

1. Algoritma tanda tangan digital yang digunakan dapat diganti dengan algoritma yang dipakai Bitcoin saat ini (ECDSA: *Elliptic Curve Digital Signature Algorithm)* yang lebih aman.
2. Pada algoritma, skema *proof of work* dapat diganti dengan *proof of stake*. Pada algoritma ini, *verificator* mempertaruhkan sejumlah koinnya supaya bloknya diterima (jika bloknya diterima, ia memperoleh sejumlah koin yang dipertaruhkan, jika bloknya memuat transaksi yang tidak valid, ia akan kehilangan sejumlah koin tersebut).
3. Algoritma yang sudah dibuat dapat di satukan sehingga terintegrasi secara otomatis.

DAFTAR PUSTAKA

Blake, I.F., dan T. Garefalakis. 2002. "On the Security of Digital Signature Algorithm." *Designs, Codes and Cryptography* 87-96.

Champagne, Phill. 2014. *The Book of Satoshi.* e53 Publishing LLC.

Goodin, Dan. 2019. *New crypto-cracking record reached, with less help than usual from Moore's Law*. 3 December. https://arstechnica.com/information-technology/2019/12/new-crypto-cracking-record-reached-with-less-help-than-usual-from-moores-law/.

Hoffstein, Jeffrey, Jill Pipher, dan Joseph H. Silverman. 2014. *An Introduction to Mathematical Cryptography.* New Yorks: Springer-Verlag.

Krause, Max J., dan Thabet Tolaymat. 2018. "Quantification of energy and carbon costs for mining cryptocurrencies." *Nature Sustainability* 711–718.

Monetha. 2019. *What Are Unspent Transaction Outputs?* 24 May. Diakses May 3, 2020. https://medium.com/@monetha/what-are-unspent-transaction-outputs-e483b7e2781f.

Rollins, Montgomery. 2003. *Money and Investments.* Kessinger Publishing, LLC.

Saif Al-Kuwari, James H. Davenport, Russell J. Bradford. 2011. "Cryptographic Hash Functions: Recent Design Trends and Security Notions." *Cryptology ePrint Archive.*

Sukirno, Sadono. 2016. *Makroekonomi Teori Pengantar*. Jakarta: Rajawali Pers.

2020. *Wallstreetmojo.* Diakses Mei 3, 2020. https://www.wallstreetmojo.com/bitcoin-vs-blockchain/.