Penggunaan *Storable Proof of Work* Untuk Mencegah *Distributed Denial of Service* (DDoS)

Hanif Arroisi Mukhlis[1], Rinaldi Munir[2],

[1,2]*Program Studi Teknik Informatika,*

Sekolah Teknik Elektro dan Informatika (STEI), Institut Teknologi Bandung

[1]hanif.arroisi@gmail.com

[2]rinaldi@informatika.org

"

Abstrak

Serangan DDoS menjadi ancaman utama *availability server*. Serangan ini bekerja dengan menghabiskan kapasitas *server*, sehingga pengguna tidak dapat mengakses *server*. *Proof of work* dapat digunakan untuk menangkal DDoS dengan memaksa penyerang melakukan perhitungan sulit dalam jumlah besar. Kelemahan *proof of work* adalah *client* harus melakukan keseluruhan protokol untuk setiap koneksi, sehingga mengakibatkan *lag*.

Storable proof of work mengembangkan *proof of work* klasik dengan membuat bukti kerja yang dapat disimpan di awal waktu, sehingga ketika dibutuhkan, bukti kerja dapat digunakan secara langsung. *Storable proof of work* menggunakan enkripsi kunci simetrik dan *message authentication code* untuk menjamin keamanan protokol. Bukti kerja memiliki waktu kadaluarsa sehingga penyerang tidak dapat menyimpan bukti kerja dalam jumlah besar. Untuk menjamin keamanan, *storable proof of work* membutuhkan dua entitas eksternal yaitu *initiator* dan *minter* yang dipercaya oleh *server*. *Storable proof of work* dapat dikembangkan lebih lanjut untuk menambah fitur seperti *third party verifier* dan data berasosiasi.

Implementasi *storable proof of work* dibuat menggunakan bahasa pemrograman Rust dan berbagai *library* kriptografi eksternal. Implementasi diuji menggunakan sebuah *script* rangkaian pengujian yang dijalankan pada satu komputer *low end* dan satu komputer *high end*. Pengujian *storable proof of work* dilakukan untuk mencari parameter optimum dan efek algoritma-algoritma di dalam *storable proof of work* terhadap pembangkitan bukti kerja. Salah satu hasil pengujian adalah persamaan target waktu pembangkitan bukti kerja terhadap parameter tingkat kesulitan.

Berdasarkan pengujian yang dilakukan, protokol *storable proof of work* dapat menangkal serangan DDoS dan menghapus *lag*. *Storable proof of work* didesain agar tahan berbagai ancaman seperti *pre-mining*. Tingkat kesulitan bukti kerja dapat diatur agar menyulitkan penyerang tetapi tidak membebani pengguna. Ketahanan *storable proof of work* dapat ditingkatkan lebih baik dengan pemilihan algoritma bukti kerja yang baik.

Kata kunci: DDoS, *storable proof of work*, kriptografi, protokol, *cyber attack*

Abstract

DDoS attack is a major threat to server performance. This attack works by consuming server capacity, so users cannot access the server. Proof of work can be used to ward off DDoS by forcing attackers to perform large amounts of difficult calculations. The downside of proof of work is that the client must perform the entire protocol for each connection, resulting in lag.

Storable proof of work develops classic proof of work by creating proof of work that can be stored early, whenever needed it can be used immediately. The storable proof of work uses symmetric encryption keys and message authentication codes to ensure the security of the protocol. Proof of work has an expiration date so attackers cannot store large amounts of proof of work. To ensure security, storable proof of work requires two external entities, the initiator and minter, which are trusted by the server. The storable proof of work can be further developed to add features such as third-party verification and data association.

The storable proof-of-work implementation was created using the Rust programming language and various external cryptographic libraries. The implementation was tested using test suite scripts run on one low end computer and one high end computer. The storable proof of work test was carried out to find the optimum parameters and the effect of the

algorithms in the storable proof of work on the generation of proof of work. One of the test results is the equation of the target time for generating proof of work with the difficulty level parameters.

Based on tests carried out, the storable proof of work protocol can ward off DDoS attacks and eliminate lag. Storable proof of work is designed to withstand threats such as pre-mining. The difficulty level of proof of work can be set to make it difficult for attackers but not overwhelming for users. The robustness of the storable proof of work can be further improved by selecting a good proof of work algorithm.

Keywords: *DDoS, Storable Proof of Work, cryptography, protocol, cyber attacks*

"

I. PENDAHULUAN

Pengguna internet menurut Survei Sosial Ekonomi Nasional (Susenas) yang diselenggarakan oleh Badan Pusat Statistik mencatat perkembangan yang sangat pesat dari 39, 90% pada tahun 2018 menjadi 66,48% pada tahun 2022 [1][2]. Dampak perkembangan teknologi dan percepatan informasi ini juga diiringi dengan berkembangnya kegiatan *cyber* crime, penyebaran *malware,* maupun *Hacktivism* yang salah satu akibatnya adalah kebocoran data ke publik. Salah satunya adalah melalui serangan *Distributed Denial of Services* (DDoS) [3] [4] [5]. Berdasarkan reviu, dinyatakan bahwa kegiatan *Hacktivism* terus merajalela sepanjang tahun 2022 dan *Microsoft* memitigasi rata-rata 1.435 serangan DDoS per hari [6].

Proof of Work (PoW) [7] [8] merupakan salah satu bentuk pertahanan yang dinilai cukup efektif untuk mencegah serangan DDoS. Kelebihan PoW adalah memiliki keamanan tinggi, terbukti berhasil karena telah digunakan secara luas dalam *blockchain* seperti Bitcoin dan Ethereum, dan memiliki sistem desentralisasi jaringan.

Penggunaan PoW untuk keamanan rekam medis [9] dilakukan dengan menggunakan *Private Blockchain* dan konsensus PoW untuk memproses data ke dalam *blockchain*. SHA-256 [10][11], yang merupakan algoritma kriptografi asimetris digunakan untuk menghasilkan sepasang *encryption key* yang diberikan kepada pengguna sebagai token untuk mengakses data yang didistribusikan.

PoW menangkal simulasi serangan DoS sehingga *client* menjadi kesulitan untuk menghabiskan *resource server* [12].

Kelemahan utama algoritma *proof-of-work* standar adalah perhitungan bukti kerja setiap kali melakukan koneksi. Hal ini menimbulkan latensi, yang diinterpretasikan oleh pengguna sebagai *lag*. Untuk mengatasi hal tersebut, *Privacy Pass* [13] menambah mekanisme menukar bukti kerja menjadi tiket untuk digunakan *client* ketika akan membuat koneksi. Sedangkan TOR [14] mengatur tingkat kesulitan bukti kerja secara dinamik. Sistem dapat meningkatkan tingkat kesulitan saat keadaan stress, seperti saat serangan DDoS terjadi, dan menurunkannya pada kondisi normal. Kedua mekanisme tersebut masih memiliki kelemahan karena tiket yang dapat digunakan ulang akan menurunkan *privacy* dan *lag* yang masih ada ketika sistem sedang diserang. Chakraborty et. al [15] menggunakan AI untuk mengatur kesulitan secara dinamis, sehingga memitigasi kelemahan tetapi belum sepenuhnya mengatasi *lag*.

Untuk mengatasi hal tersebut, dilakukan penelitian dengan menggunakan *Storable Proof of Work* untuk mencegah terjadinya serangan DDoS pada jaringan. *Storable* PoW dinilai dapat menyimpan data pengguna sah yang pernah mengakses jaringan, sehingga mengurangi kebutuhan energi PoW karena pengguna tidak perlu menyelesaikan teka-teki untuk masuk ke jaringan. Hal ini berguna untuk mengurangi latensi saat terdapat banyak pengguna yang bergabung dalam jaringan. Dengan adanya *Storable* PoW, konsumsi energi tinggi dibutuhkan untuk memverifikasi pengguna baru sebagai bentuk pertahanan terhadap serangan DDoS.

Penelitian ini bertujuan untuk membuat protokol *storable proof of work* dimana bukti kerja dapat disimpan dan digunakan di lain waktu, menguji protokol pada simulasi sistem *client-server* dan menguji protokol pada simulasi penyerang dengan *resource* terbatas.

II. ANALISIS DAN PERANCANGAN SOLUSI

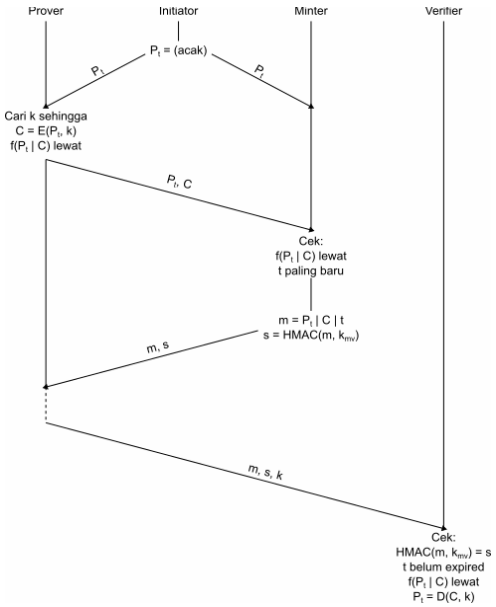
Storable proof of work memisahkan pembuktian kerja dan pengecekan kebenaran bukti kerja. Hasil bukti kerja kemudian dapat disimpan. Ketika bukti kerja akan digunakan, dilakukan pengecekan bukti kerja.

Untuk melakukan hal tersebut, diperlukan suatu relasi yang menghubungkan bukti kerja (yang publik), dan kunci rahasia. Dipilih algoritma kunci simetrik sebagai relasi. Pemetaan dilakukan antara pasangan *plaintext-ciphertext* ke kuncinya. Bukti yang dibuat dengan pasangan *plaintext-ciphertext* dapat dihubungkan ke kunci simetrik yang pembukti tahu. Hanya mengetahui pasangan *plaintext-ciphertext* tidak dapat menemukan kuncinya karena jaminan kunci simetrik.

Untuk mencegah serangan *pre-mining*, bukti kerja memiliki waktu kadaluarsa. Setelah waktu tersebut,

bukti tidak akan diterima. Hal ini penting untuk mencegah penyerang menyimpan ratusan atau lebih bukti kerja, siap untuk menyerang secara bersamaan. Agar bukti mempunyai kadaluarsa, perlu cara menandai bukti dibuat di suatu waktu. Tanda tersebut harus dipastikan otentitasnya, jika tidak bukti lawas dapat ditandai ulang sebagai bukti baru. Algoritma HMAC dipilih sebagai penjamin otentitas bukti dan adanya entitas tambahan yaitu *initiator* dan *minter.*

Diagram protokol storable proof of work sebagaimana Gambar 1. berikut ini:



Gambar 1. Diagram protokol proof of work

Berdasarkan Gambar 1, terdapat 4 pihak/entitas di dalam protokol *storable proof of work,* yaitu *prover (client)*, *initiator*, *minter*, dan *verifier (server)*. *Initiator*, *minter*, dan *verifier* diasumsikan saling mengenal/terpercaya. Komunikasi dilakukan menggunakan *secure channel*, seperti TLS.

Langkah-langkah protokol *proof of work* sebagai berikut:

1. Secara periodik, *initiator* membuat blok *plaintext* acak P_t (dengan *t* periode saat *plaintext* dibuat).
2. *Prover* mengenkripsi *plaintext* menggunakan kunci enkripsi acak, menghasilkan pasangan *plaintext-ciphertext*.

$$C = E(P_t, k) \quad (1)$$

3. *Prover* memeriksa pasangan *plaintext-ciphertext* menggunakan suatu fungsi $f(P_t|C)$ yang sulit untuk dipecahkan (untuk membuktikan kerja). Jika fungsi menghasilkan nilai *false*, *prover* kembali ke langkah 2, mencoba lagi dengan kunci enkripsi lain.
4. *Prover* mengirim (P_t, C) ke *minter*.
5. *Minter* memeriksa *plaintext* berasal dari blok *plaintext* yang dibuat oleh *initiator*, *t* adalah periode sekarang, dan $f(P_t|C)$ menghasilkan nilai *true*.
6. Jika sukses, *minter* menambahkan kode integritas menggunakan HMAC (3) dan mengirim balik pesan (m, h) ke *prover*.

$$m = (P_t, C, t) \quad (2)$$
$$h = HMAC(m, k_{mv}) \quad (3)$$

Dengan k_{mv} adalah kunci rahasia yang diketahui oleh *minter* dan *verifier*.

7. Setelah beberapa waktu, *prover* mengirim (m, h, k) ke *verifier*.

8. *Verifier* memeriksa bahwa:
   a. Bukti kerja belum pernah digunakan sebelumnya.
   b. Kode integritas valid.
   c. *Timestamp* belum kadaluarsa.
   d. Kunci mendekripsi *ciphertext* kembali ke *plaintext*. (4)

$$P_t = D(C, k) \quad (4)$$

Varian protokol menggunakan *signature scheme* hampir sama, dengan HMAC diganti dengan *signature*. Kunci privat s_k dipegang oleh *minter*, dan kunci publik p_k dipegang oleh *verifier*.

Minter: $s = Sign(m, s_k)$ (5)
Verifier: $Verify(m, s, p_k)$ (6)

Contoh kalkulasi protokol adalah sebagai berikut:

a. *Plaintext* :
`DAC1C8248D127785D86B273EE65555A6`
b. *Encryption Key* :
`C1A083F574FBB3ED7E8ED73F2FF72F09`
c. *Ciphertext* :
`6EB1AFDD2E2C1E6F1FB61667D98F4CF7`
d. $SHA256(P|C)$:
`FF7D943FBDB9B214088AD1FFBBC7F14F00510`
`11A041E7D0E9D0D66637EA121BC`
e. *Timestamp* :
`6C2E0400C12AF7ED`
f. HMAC *Key* :
`DC935D3BD3CAB783BF5ED7CF37DA2EDA532A3`
`B9965CBAB35CC26C4CBEE1EC670`
`DBBE294E8D187D00B36BEC6AA80F833904E69`
`455FEB4E156697DD56676A42759`

g. $HMAC(P|C|t,k)$:
`835CE5A0CBA7D00C401B80887D9318A633052`
`5776C7C386A0712948085CF5263`

III. IMPELEMENTASI DAN HASIL PENGUJIAN

A. *Implementasi Prototype Protokol Storable Proof of Work*

Implementasi dan pengujian dilakukan pada dua *hardware*, merepresentasikan komputer *low-end* dan *high-end*. *Software* untuk membuat implementasi adalah bahasa pemrograman *Rust*, kompiler standar *rustc* (dengan *backend compiler* LLVM), dan *package manager Cargo*.

Tabel 1. Spesifikasi sistem

Spesifikasi	Sistem 1 (*low-end*)	Sistem 2 (*high-end*)
Operating System	Windows 10 22H2	Windows 11 23H2
Processor	Intel Core i5-8250U	Intel Core i7-11800H
Core Count	4 *core* (8 *thread*)	8 *core* (16 *thread*)
Clock Speed	1.8 GHz	2.3 GHz
Memory	DDR3 12GB 2400MHz	DDR4 16GB

Prototipe diimplementasikan di dalam *crate/module* `spow-core` menggunakan tipe data generik. Hal ini dilakukan untuk memudahkan mengganti algoritma, seperti fungsi *hash*, enkripsi, skema *signature*, atau bukti kerja. Didefinisikan empat tipe data mensimulasikan *initiator*, *prover*, *minter*, dan *verifier*. Tipe data adapter menspesialisasikan untuk kelas algoritma tertentu. Definisi setiap tipe data dan kegunaannya terdapat pada Tabel 2.

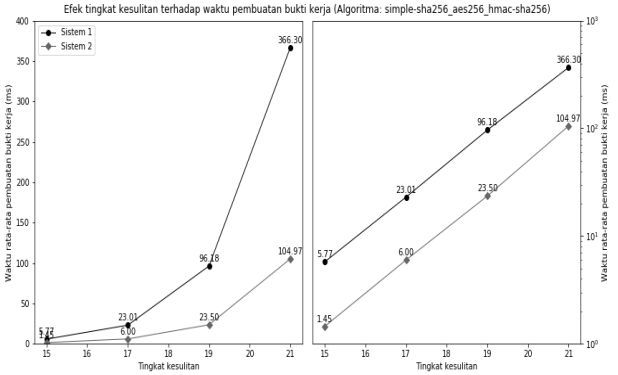
Tabel 2. Struktur tipe data

Tipe Data	Kegunaan
`Initiator<R>`	Mensimulasikan *initiator*. Menghasilkan blok-blok *plaintext* menggunakan *random number generator*.
`Prover<Cipher, Difficulty>`	Mensimulasikan *prover*. Membuat bukti kerja menggunakan *cipher* dan fungsi *difficulty*/bukti kerja.
`BlockCipherProver<Cipher>`	Adapter *prover* untuk *block cipher*. Melakukan enkripsi/dekripsi menggunakan kunci acak.
`StreamCipherProver<Cipher>`	Adapter *prover* untuk *stream cipher*. Melakukan enkripsi/dekripsi menggunakan kunci dan *initialization vector* acak.
`Minter<Inner, Difficulty>`	Mensimulasikan *minter*. Memverifikasi kerja, membubuhi *timestamp*, dan menandatangani bukti kerja.
`HmacMinter<Hash, Key: ArrayLength<u8>>`	Adapter *minter* untuk HMAC menggunakan fungsi *hash* dan kunci rahasia.
`SignMinter<Sign, Digest, Output>`	Adapter *minter* menggunakan *signature scheme* dan fungsi *digest/hash*.
`RandomSignMinter<Sign, Random, Digest, Output>`	Adapter *minter* menggunakan *probabilistic signature scheme*. Memerlukan *cryptographically secure random number generator* untuk menjamin keamanan.
`Verifier<Inner, Cipher, Difficulty>`	Mensimulasikan *verifier*. Menerima pesan bertanda tangan dan kunci rahasia, kemudian memverifikasi dan mencoba dekripsi *ciphertext*.
`HmacVerifier<Hash, Key: ArrayLength<u8>>`	Adapter *verifier* menggunakan HMAC untuk memverifikasi integritas pesan. Membutuhkan kunci rahasia yang sama dengan *minter*.
`SignVerifier<Verifier, Digest, Output>`	Adapter *verifier* menggunakan *signature scheme* untuk memverifikasi tanda tangan.

B. *Pengujian Storable Proof of Work*

Pengujian dilakukan dengan melakukan serangkaian tes. Setiap tes dilakukan dengan 2 periode 2,5 menit (waktu total 5 menit). Rangkaian selalu dimulai dengan 5 menit *stress test* untuk memastikan sistem telah mencapai ekuilibrium. Nilai keluaran tes adalah waktu rata-rata pembangkitan bukti kerja.

Digunakan 2 jenis sistem/komputer untuk melakukan pengujian. Sistem 1 merepresentasikan komputer dengan kemampuan terbatas, dan sistem 2 merepresentasikan komputer berspesifikasi tinggi.

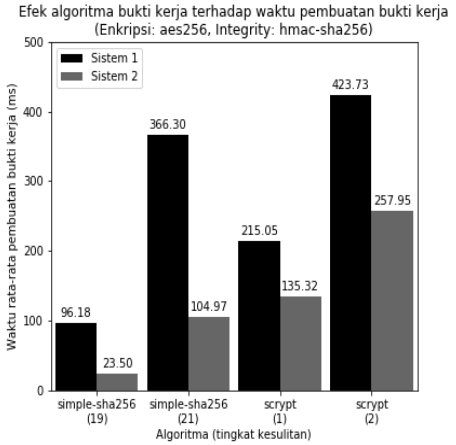


Gambar 2. Efek tingkat kesulitan

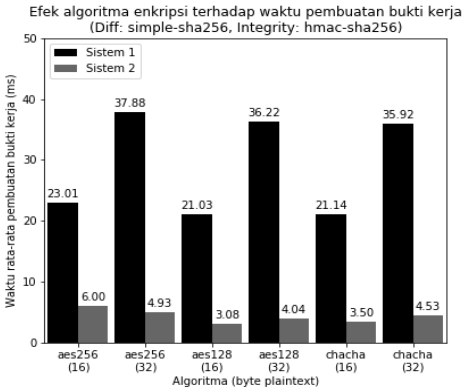
Dari gambar 2, didapat waktu rata-rata pembangkitan bukti kerja meningkat seiring dengan tingkat kesulitan. Kurva berbentuk eksponensial

karena tingkat kesulitan diukur berdasarkan *n*-bit yang bernilai 0. Dengan asumsi setiap bit mempunyai probabilitas ½ bernilai 0, maka dapat diprediksi kurva berbentuk eksponensial dengan basis eksponen 2 ($y = a2^x$). Melakukan regresi eksponensial pada kurva sistem 1, didapat koefisien $a = 0.0001742445013084277$ dan $b = 2.001995532937$, sesuai dengan prediksi.

Gambar 2. juga menunjukkan sistem 2 kurang lebih memiliki performa ~4x dibandingkan sistem 1 (terlihat lebih jelas di kurva logaritmik di sebelah kanan). Hal ini tidak selalu benar untuk semua kombinasi varian algoritma, seperti yang ditunjukkan pada Gambar 3.



Gambar 3. Efek algoritma bukti kerja



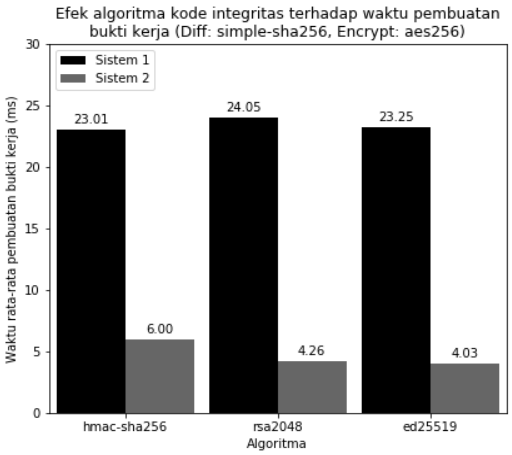
Gambar 4. Efek algoritma enkripsi

Gambar 3. menunjukkan perbedaan jenis algoritma bukti kerja mempengaruhi kesulitan bukti kerja secara signifikan. Algoritma *scrypt* didesain untuk *password hashing* (walaupun dapat dimodifikasi untuk *proof-of-work*). Hanya dengan tingkat kesulitan 1, waktu pembuatan bukti kerja menggunakan *scrypt* 2 kali lipat lebih sulit daripada SHA-256 pada tingkat kesulitan 19. Karena tingkat kesulitan tidak bisa lebih kecil daripada 1, algoritma *scrypt* tanpa modifikasi kurang cocok untuk membuat bukti kerja dengan cepat.

Sistem 2 menunjukkan potensi keuntungan menggunakan *scrypt*. Karena *scrypt* dirancang untuk mengurangi perbedaan kecepatan CPU, perbedaan waktu antara sistem 1 dan sistem 2 lebih kecil dibandingkan dengan SHA-256.

Gambar 4. menunjukkan efek algoritma enkripsi terhadap waktu pembangkitan bukti kerja kecil. Algoritma AES-256 sedikit lebih berat dibandingkan AES-128, mungkin karena kunci lebih besar. Meningkatkan ukuran *plaintext* meningkatkan kesulitan secara linear. Digabungkan dengan tingkat kesulitan, kedua variabel dapat diatur untuk mengatur kesulitan bukti kerja secara presisi.

Perbedaan waktu pada sistem 2 lebih kecil, dan efek meningkatkan panjang *plaintext* kurang berefek. Kemungkinan hal ini terjadi karena *bottleneck* bukan pada proses enkripsi.



Gambar 5. Efek algoritma *integrity code*

Gambar 5. menunjukkan algoritma *integrity code* tidak mempengaruhi waktu pembangkitan bukti kerja secara signifikan. Perbedaan waktu terjadi karena simulasi dilakukan *end-to-end*, dari *initiator* hingga *verifier*. Sehingga efek dari *minter* dan *verifier* akan menambah waktu pembuatan bukti kerja, walaupun sedikit karena hanya jumlah bukti yang dibuat kecil dibandingkan jumlah kandidat bukti kerja yang dibangkitkan.

Setiap algoritma memiliki kelebihan dan kekurangan sendiri. Algoritma *public-key signature* (RSA dan Ed25519) memisahkan kunci untuk *minter* dan *verifier*, memungkinkan kegunaan seperti *verifier* pihak ketiga. HMAC pada dasarnya tahan algoritma kuantum, dan membutuhkan asumsi kriptografi lebih sedikit dibandingkan dengan algoritma *public-key*.

C. Analisis Hasil Pengujian

Dari pengujian dapat disimpulkan bahwa pemilihan algoritma enkripsi dan *integrity code* memiliki efek kecil terhadap bukti kerja. Hal ini menguntungkan, karena algoritma-algoritma tersebut dapat memberikan fitur tambahan ke protokol *storable proof-of-work*, seperti *verifier* pihak ketiga, atau data berasosiasi/*associated data*. Panjang *plaintext* dapat mempengaruhi waktu pembangkitan, namun efeknya kurang reliabel.

Pemilihan algoritma *difficulty/*bukti kerja menjadi titik kritikal dari protokol. Menggunakan algoritma yang terlalu sederhana (simple SHA-256) dapat memberikan penyerang dengan komputasi paralel keuntungan besar, sedangkan algoritma *password-hashing* seperti *scrypt* dapat mengurangi perbedaan kemampuan komputer. Karena kebutuhan dunia nyata berbeda-beda, parameter tingkat kesulitan *n* akan berbeda juga. Seperti kebutuhan untuk *web* membutuhkan latensi sub-milisekon, sedangkan kebutuhan lain dapat mentoleransi hingga beberapa menit.

Dari Gambar 2. dapat disimpulkan bahwa perhitungan paling berat adalah membangkitkan bukti kerja. Karena parameter tingkat kesulitan hanya mempengaruhi probabilitas sukses kandidat bukti kerja dengan nilai 2^{-n} dan eksponen kurva mendekati 2.

Karena setiap *server* memiliki kapasitas berbeda-beda, pengaturan *storable proof of work* perlu disesuaikan dengan kapasitas *server*. Waktu pembangkitan bukti kerja adalah invers dari *request* per detik. Waktu kadaluarsa dibagi dengan waktu pembangkitan bukti kerja menjadi jumlah *request* maksimum dalam waktu bersamaan. Sebagai contoh, jika *server* dapat memproses 10 *request per second* dan 600 *request* bersamaan untuk satu pengguna/penyerang, maka target waktu pembangkitan bukti kerja adalah 100 milisekon dan waktu kadaluarsa adalah 1 menit.

Pada Tabel 4. diberikan nilai rekomendasi tingkat kesulitan untuk target waktu pembangkitan bukti kerja. Perhitungan dilakukan dengan melakukan regresi eksponensial kurva pada Gambar 2. untuk kedua sistem (7). Untuk setiap target waktu dihitung nilai *n*, dibulatkan ke bilangan bulat terdekat (8). Karena *n* lebih kecil dari 1 tidak bermakna, pada Tabel 4. dinotasikan sebagai <1. Karena kurangnya titik data untuk *scrypt*, perhitungan menggunakan ekstrapolasi dari kurva simple SHA-256, dengan asumsi kedua kurva memiliki bentuk yang hampir sama, hanya berbeda skala. Estimasi nilai *n* untuk algoritma *scrypt* dilakukan dengan menggunakan rasio antara waktu pembangkitan menggunakan algoritma SHA-256 dan *scrypt* untuk n = 1 (9 dan 10). Semua hasil perhitungan konstanta terdapat pada Tabel 4.3.

$$t(n) = ab^n \quad (7)$$

$$n(t) = \left\lceil \frac{log(t) - log(a)}{log(b)} \right\rceil \quad (8)$$

$$r = \frac{t\prime(1)}{t(1)} \quad (9)$$

$$n'(t) = \left\lceil \frac{log(t) - log(a) - log(r)}{log(b)} \right\rceil \quad (10)$$

Tabel 3. Struktur tipe data

Konstanta	Nilai (Sistem 1)	Nilai (Sistem 2)
a	1.742445e-04	3.395801e-05
b	2.001996e+00	2.034484e+00
r	6.164883e+05	1.958660e+06

Tabel 4. Rekomendasi nilai tingkat kesulitan (*n*)

Waktu Pembangkitan Bukti Kerja	Nilai *n* Sistem 1 (SHA-256)	Nilai *n* Sistem 2 (SHA-256)	Nilai *n* Sistem 1 (*scrypt*)	Nilai *n* Sistem 2 (*scrypt*)
100 milisekon	19	21	<1	1
1 sekon	22	24	3	4
1 menit	28	30	9	10
10 menit	32	33	12	13
1 jam	34	36	15	15

IV. SIMPULAN

Berdasarkan analisis dan pengujian dapat disimpulkan bahwa:

1. Protokol *storable proof-of-work* dapat digunakan untuk sistem yang memerlukan menyimpan bukti kerja. Bukti kerja dapat disimpan selama waktu terbatas dan tidak dapat dimanipulasi, sehingga hanya dapat dibangkitkan sebanyak tertentu.
2. *Storable proof-of-work* dapat memverifikasi integritas bukti kerja menggunakan HMAC atau *public-key signature*.
3. *Storable proof-of-work* jika dikonfigurasi dengan tepat dapat menangkal serangan DDoS dengan membatasi kecepatan penggunaan *resource*.
4. Algoritma di dalam *storable proof-of-work* dapat menambah fitur seperti verifikasi *third-party* dan data berasosiasi.
5. Proses terberat dari *storable proof-of-work* adalah proses pembangkitan bukti kerja. Tingkat kesulitan dapat diatur sehingga cukup ringan untuk dijalankan pada sistem *low-end*.
6. Pemilihan algoritma bukti kerja yang tepat dapat mengurangi keuntungan penyerang tanpa membebani pengguna terlalu berat.

REFERENSI

[1] Badan Pusat Statistik Indonesia, *Statistik telekomunikasi Indonesia 2022*. Retrieved from:https://www.bps.go.id/id/publication/2023/08/31/131385d0253c6aae7c7a59fa/statistik-telekomunikasi-indonesia-2022.htmlBadan Pusat Statistik

[2] Asosiasi Penyelenggara Jasa Internet Indonesia (APJII). (2024, February 12). *Survei APJII pengguna internet di Indonesia tembus 215 juta orang*. Retrieved from APJII: https://apjii.or.id/berita/d/survei-apjii-pengguna-internet-di-indonesia-tembus-215-juta-orang

[3] Robert, M. S. (2022). Computer viruses. In *Encyclopedia of information systems*. Elsevier (pp.255-265)

[4] Gurcok, C. (2017). Securing cloud computing system. In Vacca, J. R., *Computer and information security handbook (third edition)*. Elsevier (pp. 897-922)

[5] O\'Brien, J. A., Marakas, G. M. (2010). *Introduction to information systems*. 15th ed. New York: McGraw-Hill Irwin

[6] Microsoft. (2023). *2022 in review DDOS attack trends and insights*. Retrieved from: Microsoft corporation: https://www.microsoft.com/en-us/security/blog/2023/02/21/2022-in-review-ddos-attack-trends-and-insights/

[7] Ethereum. (2024, February 12). *Proof-of-work (POW)*. Retrieved from: https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/ archive/macros/latex/contrib/supported/IEEEtran/

[8] Getmonero. (2024, February 12). *About monero: a brief history*. Retrieved from: https://www.getmonero.org/resources/about/

[9] Wikarsa, L., Suwanto, T. & Lengkey, C. (2022), "Implementasi algoritma konsensus Proof-of-Work dalam Blockchain terhadap rekam medis. Jurnal Pekommas", 7, 1 (Jun. 2022), 41–52. https://doi.org/10.56873/jpkm.v7i1.4403

[10] Watkins, D. (2014, May). *Scrypt mining with ASICs*. Retrieved from: https://www.researchgate.net/publication/361039194_Scrypt_Mining_with_ASICs

[11] Yoshida, H., & Biryukov, A. (2005, August). *Analysis of SHA-256 variant*. Retrieved from: https://www.researchgate.net/publication/221274799_Analysis_of_a_SHA-256_Variant

[12] Lundqvist, A., Landsberg, J. (2012). *DoS mitigation using proof of work*. CSC KTH.

[13] Privacy Pass [Online], https://privacypass.github.io/protocol/

[14] TOR [Online], https://www.torproject.org/about/history/

[15] Chakraborty, T., Shaswata, M., Mittal, S., Young, M. (2022). *AI adaptive PoW: an AI assisted proof of work (PoW) framework for DDOS defense*. Software Impacts 13 (2022) 100335. https://doi.org/10.1016/j.simpa.2022.100335