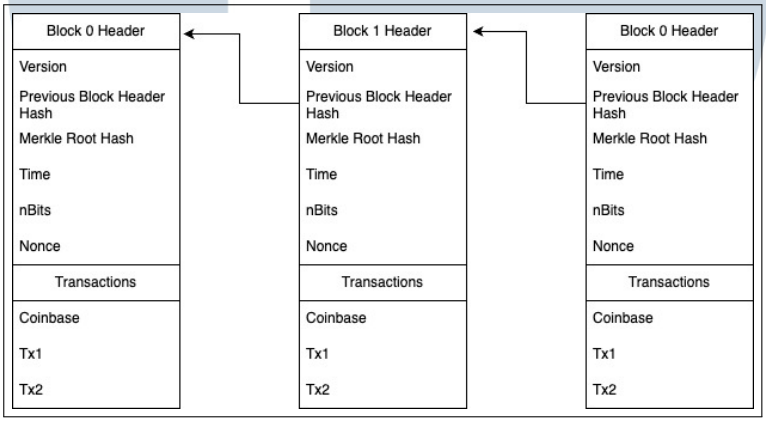
BAB 2
LANDASAN TEORI

2.1 Blockchain

Blockchain adalah sebuah buku besar digital yang menyimpan data-data transaksi dalam sebuah *block* sehingga membentuk sebuah sejarah transaksi yang lengkap [9]. Setiap *block* pada blockchain saling terhubung sehingga membentuk struktur layaknya *chain* atau rantai. Struktur dari blockchain dapat dilihat pada Gambar 2.1.



Gambar 2.1. Struktur data pada blockchain [10]

Salah satu kelebihan utama dari blockchain adalah *decentralization*, yaitu kemampuan dari sistem tersebut untuk tidak bergantung pada pihak ketiga ataupun pihak terpusat [11]. Seluruh keputusan pada blockchain ditentukan oleh para peserta dalam jaringan blockchain itu sendiri. Mekanisme konsensus digunakan pada blockchain agar seluruh peserta pada jaringan blockchain mencapai kesepakatan terhadap *state* dari data yang tersimpan [9]. Setelah itu, data pada blockchain tidak hanya disimpan pada satu tempat, namun seluruh peserta pada jaringan blockchain akan menyimpan salinan data yang sama sehingga terjadinya manipulasi data pada blockchain dapat dikatakan hampir mustahil. Selain itu, integritas dari blockchain juga dapat terjaga karena setiap *block* menyimpan data *hash* dari dirinya sendiri dan *block* sebelumnya. Dengan hal ini, manipulasi data dapat dicegah karena ketika data pada suatu *block* berubah, maka *hash* dari seluruh *block* berikutnya akan ikut berubah.

Bitcoin merupakan generasi pertama dari blockchain yang diperkenalkan oleh Satoshi Nakamoto pada tahun 2009 yang berfokus pada sistem pembayaran dan mata uang digital berbasis kriptografi [12]. Setelah itu, generasi blockchain berikutnya diperkenalkan oleh Ethereum yang memberikan fungsionalitas *smart contract* pada blockchain, yaitu sebuah kode atau program yang disimpan pada jaringan blockchain yang merepresentasikan versi digital dari kontrak tradisional [13]. Dengan *smart contract*, para pengembang dapat membangun aplikasi yang dieksekusi secara *decentralized* pada jaringan blockchain Ethereum, misalnya NFT, mata uang, dan berbagai macam layanan keuangan.

Pada dasarnya, Ethereum adalah sebuah protokol blockchain yang berfungsi untuk membangun *decentralized application* [14]. Dalam protokol Ethereum, terdapat dua jenis akun, yaitu *externally owned account* (EOA) dan *contract account*. Kedua akun tersebut terdiri dari *private key* dan juga *public key* yang dihasilkan oleh algoritma *elliptic curve digital signature algorithm* (ECDSA) [15]. Pada EOA, *private key* dari akun tersebut dimiliki dan dikontrol oleh pengguna. Sedangkan pada *contract account*, *private key* tidak dimiliki oleh siapapun, sehingga *contract account* hanya dapat dikontrol berdasarkan kode yang tertulis di dalamnya. Pada blockchain yang menerapkan protokol Ethereum, EOA dapat melakukan dan memulai transaksi, yaitu proses menandatangani sebuah paket data secara digital sebagai pesan untuk dikirim ke akun lainnya [14].

Protokol Ethereum memiliki mekanisme yang dikenal sebagai *gas*, yaitu sebuah satuan yang mengukur berapa besar upaya komputasi untuk melakukan operasi tertentu pada jaringan Ethereum [16]. Untuk mengeksekusi sebuah transaksi pada Ethereum, dibutuhkan sumber daya komputasi sehingga perlu ada biaya (*fee*) yang dibayarkan pada pihak yang memelihara jaringan tersebut, yaitu *miner*. Biaya tersebut (*gas fee*) dapat disebut sebagai *gas fee* dan dibayarkan dengan menggunakan ether (ETH), yaitu *native currency* dari protokol Ethereum. Jumlah *gas fee* yang perlu dibayarkan dapat dihitung berdasarkan rumus berikut [16].

$$gas\,fee = gas\,used \times (base\,fee + priority\,fee) \tag{2.1}$$

Pada rumus 2.1, *gas used* merupakan jumlah *gas* yang digunakan untuk melakukan sebuah operasi atau transaksi pada jaringan Ethereum. *Base fee* adalah biaya minimum (dalam satuan GWEI) yang perlu dibayar untuk setiap unit *gas* yang digunakan agar transaksi dapat dimasukkan ke blok pada blockchain. *Priority fee* adalah biaya tambahan atau *tip* yang diberikan pada *miner* agar transaksi yang

dilakukan dapat lebih diprioritaskan oleh *miner*. Selain untuk membayar *miner* atas sumber daya komputasi yang digunakan, *gas fee* juga bertujuan untuk menjaga keamanan jaringan Ethereum, misalnya mencegah terjadinya *spamming* oleh pihak tidak bertanggung jawab pada jaringan [16].

2.2 ZK-SNARK

ZK-SNARK merupakan salah satu jenis dari *zero-knowledge proof*. *Zero-knowledge proof* adalah sebuah cara yang digunakan untuk membuktikan validitas dari sebuah *statement* tanpa harus mengungkapkan isi dari *statement* tersebut [5]. Dalam *zero-knowledge proof*, terdapat dua pihak yang terlibat, yaitu *prover* sebagai pihak yang ingin membuktikan sebuah *statement* dan *verifier* sebagai pihak melakukan validasi terhadap *statement* tersebut. Dengan menggunakan *zero-knowledge proof*, *prover* dapat membuktikan bahwa *statement* yang diklaim benar tanpa memberikan informasi lain selain fakta bahwa *statement* tersebut benar. *Zero-knowledge proof* memiliki tiga properti utama yang harus dipenuhi [17], yaitu:

- **Completeness:** Jika *statement* benar, maka *prover* harus dapat meyakinkan *verifier*.
- **Soundness:** *Prover* yang tidak bertanggung jawab atau jahat tidak akan bisa meyakinkan *verifier* jika *statement* salah.
- **Zero-knowledge:** *Verifier* yang tidak bertanggung jawab atau jahat tidak mendapatkan informasi apapun terkait dengan *statement* selain kebenaran dari *statement* itu sendiri.

Pada mulanya, sebuah protokol *zero-knowledge proof* bersifat interaktif, yaitu terdapat proses komunikasi antara *prover* dan *verifier* yang dilakukan secara berulang untuk mencapai probabilitas kebenaran dari *proof* yang tinggi [18]. Setelah itu, Blum dkk [19] memperkenalkan *non-interactive zero-knowledge proof* yang hanya memerlukan sebuah *string* sebagai *proof* yang dikirim oleh *prover* pada *verifier*, lalu *verifier* dapat menentukan kebenaran dari *proof* tanpa adanya interaksi tambahan.

ZK-SNARK merupakan singkatan dari *Zero-Knowledge Succinct Non-Interactive Argument of Knowledge*. ZK-SNARK memiliki sifat-sifat yang sama dengan *zero-knowledge proof* pada umumnya, yaitu *completeness*, *soundness*, dan *zero-knowledge*. Meskipun demikian, ZK-SNARK memiliki dua sifat lain yang

membuatnya menjadi lebih efisien, yaitu *succinct* dan *non-interactive* [8]. *Succinct* memiliki arti bahwa *proof* yang dihasilkan oleh *prover* dapat diverifikasi dalam hitungan *millisecond* dan memiliki ukuran yang kecil, misalnya hanya ratusan bytes. Sedangkan *non-interactive* memiliki arti bahwa dalam melakukan verifikasi proof, *prover* dan *verifier* hanya berinteraksi satu kali tanpa adanya interaksi bolak-balik atau interaksi tambahan.

Ellliptic Curve Digital Signature Algorithm (ECDSA) adalah algoritma kriptografi yang digunakan dalam Ethereum untuk menghasilkan *public key* dan *private key* [20]. ECDSA dapat menghasilkan sebuah *digital signature* yang terdiri atas tiga parameter, yaitu r (32 bytes), s (32 bytes), dan v (1 byte) sehingga memiliki ukuran total hanya 65 bytes [21]. Ukuran tersebut sangat kecil, sehingga *digital signature* yang dihasilkan oleh algoritma ECDSA cocok untuk digunakan sebagai *proof* dalam ZK-SNARK.

Dalam sistem yang menggunakan skema ZK-SNARK, terdapat tiga algoritma yang digunakan di dalamnya, yaitu KeyGen(), Proof(), dan Verify() [22].

- **Keygen(C)** $\rightarrow$ {**PK, VK**}

  Keygen() adalah sebuah fungsi yang menerima C, yaitu sebuah proses sebagai parameter, lalu menghasilkan sebuah *proving key* (PK) dan *verification key* (VK).

- **Proof($\vec{w}$, $\vec{x}$, PK)** $\rightarrow \pi$

  Proof adalah sebuah fungsi yang menerima beberapa parameter, lalu menghasilkan π, yaitu *proof* yang akan digunakan untuk membuktikan *statement*. Parameter $\vec{w}$ adalah *witness* atau argumen rahasia yang hanya diketahui *prover*. Sedangkan $\vec{x}$ adalah argumen publik yang diketahui oleh *prover* dan *verifier*. Lalu PK adalah *proving key* yang dihasilkan pada algoritma Keygen().

- **Verify(π, $\vec{x}$, VK)** $\rightarrow b$

  Verify adalah sebuah fungsi yang menerima *proof* π, argumen publik $\vec{x}$, dan *verifying key* VK untuk menghasilkan *b*, yaitu *decision output* yang memiliki nilai *boolean*. Nilai *b* menentukan validitas *proof* π yang diberikan oleh *prover* pada *verifier*.

2.3 Autentikasi Identitas pada Sistem Voting

Pada sistem *voting* konvensional seperti pada Pemilu 2019 di Indonesia, pemberian suara dilakukan secara *offline* yang mengharuskan masyarakat untuk datang ke tempat pemilihan suara (TPS). Masyarakat diharuskan untuk membawa KTP dan surat C6 dan menyerahkannya pada panitia untuk diautentikasi [23]. Setelah itu, masyarakat dapat mengambil surat suara dan melakukan pencoblosan pada bilik suara.

Pada sistem voting elektronik atau *E-voting*, terdapat berbagai macam cara untuk melakukan autentikasi identitas peserta *voting*. Sistem *E-voting* berbasis web yang dibangun oleh Altun dkk [24] melakukan autentikasi identitas peserta menggunakan metode *biometric*, yaitu *fingerprint*. Sistem E-voting yang dibangun oleh Falkner dkk [25] melakukan autentikasi dengan menggunakan *pseudorandom number generator* (PRNG) untuk menghasilkan *password* yang kemudian di-*encode* menjadi sebuah *QR-Code* untuk masing-masing peserta. Sedangkan pada sistem E-voting berbasis blockchain oleh Alvi dkk [4] yang disebut sebagai DVTChain melakukan autentikasi identitas peserta dengan menggunakan *private key* pada *wallet* mereka, lalu memasukkan kredensial seperti nama, *national ID card number*, dan nomor handphone pada *smart contract* yang kemudian akan di-*hash* dan dibandingkan dengan *hash value* yang sudah tersimpan pada *smart contract* tersebut.

2.4 Throughput dan Skalabilitas

Throughput pada sebuah sistem merupakan seberapa banyak transaksi yang dapat diproses oleh sistem tersebut dalam periode waktu tertentu, salah satu istilah yang sering digunakan untuk mengukur *throughput* adalah *transaction per second* (TPS). *Throughput* memiliki hubungan yang erat dengan skalabilitas. Skalabilitas adalah kemampuan dari sebuah sistem untuk menangani beban kerja yang bertambah, baik dengan menambahkan sumber daya lebih pada sistem ataupun tidak [26]. Sebuah sistem dapat dianggap *scalable* ketika sistem tersebut dapat bekerja dengan baik dalam beban kerja yang bertambah tanpa harus melakukan perancangan ulang terhadap sistem tersebut [27]. Secara umum, terdapat dua solusi untuk meningkatkan skalabilitas dari sebuah sistem, yaitu:

- ***Vertical Scaling*:** Memaksimalkan sumber daya pada *node* untuk mengatasi beban kerja yang bertambah, misalnya dengan menambahkan *processing*

power (CPU), memori (RAM), dan komponen fisik lainnya pada *node*.

- ***Horizontal Scaling*:** Menambahkan jumlah *node* pada sistem untuk mendistribusikan beban kerja yang masuk. Melakukan hal ini juga dapat meningkatkan *availability* dari sistem dan mengurangi *single point of failure*.

Load balancing merupakan salah satu metode yang dapat digunakan untuk memastikan *availability* dan *scalability* dari sebuah sistem. Ketika melakukan *horizontal scaling* dan sistem memiliki sejumlah *node*, perlu ada sebuah sistem yang mengatur dan menentukan *node* mana yang akan merespon sebuah permintaan yang masuk agar sistem dapat melakukan eksekusi secara paralel. Sebuah *load balancer* dapat menerima permintaan dari pengguna, lalu mengarahkannya ke *node* yang tepat berdasarkan kriteria dan strategi yang digunakan [27]. Salah satu strategi yang dapat digunakan pada *load balancer* adalah *Round Robin*, yaitu setiap *node* yang tersedia akan secara bergantian merespon permintaan yang masuk.

Ketika melakukan peningkatan skalabilitas pada sebuah sistem, skalabilitas tersebut dapat diukur berdasarkan dua metrik, yaitu *speedup* dan *efficiency* [28]. *Speedup* adalah rasio waktu yang dibutuhkan untuk menjalankan sistem secara *sequential* terhadap waktu yang dibutuhkan untuk menjalankan sistem secara paralel [28]. Perhitungan *speedup* dapat dilihat pada rumus 2.2.

$$speedup = \frac{sequential\ execution\ time}{parallel\ execution\ time} \tag{2.2}$$

Sedangkan *efficiency* adalah metrik yang mengukur tingkat pemanfaatan (*utilization*) prosesor ketika sistem dijalankan dengan jumlah prosesor tertentu [28]. Perhitungan *efficiency* dapat dilihat pada rumus 2.3.

$$efficiency = \frac{speedup}{processor\ used} \tag{2.3}$$