IMPLEMENTASI FUNGSI *HASH* SHA-256 DAN KRIPTOGRAFI ALGORITMA ECDSA (*Elliptic Curve Digital Signature Algorithm*) DALAM PEMBUATAN TANDA TANGAN DIGITAL

SKRIPSI

OLEH:
ZAKIYAH NUR MILLAH
NIM. 210601110029

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2025

IMPLEMENTASI FUNGSI *HASH* SHA-256 DAN KRIPTOGRAFI ALGORITMA ECDSA (*Elliptic Curve Digital Signature Algorithm*) DALAM PEMBUATAN TANDA TANGAN DIGITAL

SKRIPSI

Diajukan Kepada
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang
untuk Memenuhi Salah Satu Persyaratan dalam
Memperoleh Gelar Sarjana Matematika (S.Mat)

Oleh
ZAKIYAH NUR MILLAH
NIM. 210601110029

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2025

IMPLEMENTASI FUNGSI *HASH* SHA-256 DAN KRIPTOGRAFI ALGORITMA ECDSA (*Elliptic Curve Digital Signature Algorithm*) DALAM PEMBUATAN TANDA TANGAN DIGITAL

SKRIPSI

Oleh
Zakiyah Nur Millah
NIM. 210601110029

Telah Disetujui Untuk Diuji

Malang, 24 Februari 2025

Dosen Pembimbing I	Dosen Pembimbing II
Muhammad Khudzaifah, M.Si	Erna Herawati, M.Pd
NIPPPK. 199005112023211029	NIPPPK. 197607232023212006

Mengetahui,
Ketua Program Studi Matematika

Dr. Elly Susanti, M.Sc
NIP. 197411292000122005

IMPLEMENTASI FUNGSI *HASH* SHA-256 DAN KRIPTOGRAFI ALGORITMA ECDSA (*Elliptic Curve Digital Signature Algorithm*) DALAM PEMBUATAN TANDA TANGAN DIGITAL

SKRIPSI

Oleh
Zakiyah Nur Millah
NIM. 210601110029

Telah Dipertahankan di Depan Dewan Penguji Skripsi
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
untuk Memperoleh Gelar Sarjana Matematika (S.Mat)

Tanggal 16 Mei 2025

Ketua Penguji	: Hisyam Fahmi, M.Kom	.....
Anggota Penguji 1	: Dr. Mohammad Jamhuri, M.Si	.....
Anggota Penguji 2	: Muhammad Khudzaifah, M.Si	.....
Anggota Penguji 3	: Erna Herawati, M.Pd	.....

Mengetahui,
Ketua Program Studi Matematika

Dr. Elly Susanti, M.Sc
NIP. 197411292000122005

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama : Zakiyah Nur Millah

NIM : 210601110029

Program Studi : Matematika

Fakultas : Sains dan Teknologi

Judul Skripsi : Implementasi Fungsi *Hash* SHA-256 dan Kriptografi Algoritma ECDSA (*Elliptic Curve Digital Signature Algorithm*) dalam Pembuatan Tanda Tangan Digital

menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan pengambilan tulisan atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka. Apabila di kemudian hari terbukti atau dapat dbuktikan skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 16 Mei 2025

Yang membuat pernyataan

Zakiyah Nur Millah

NIM. 210601110029

MOTO

"*Hidupku tenang karena mengetahui bahwa apa yang telah melewatkanku tidak akan pernah menjadi takdirku. Dan apa yang telah ditakdirkan untukku tidak akan pernah melewatkanku*"

-Umar bin Khatab

PERSEMBAHAN

Bismillahirrahmaanirrahiim

Segala puji bagi Allah SWT, Sang Maha Pengatur takdir yang telah memberikan kekuatan, kesabaran, serta ilmu yang tak terhingga dalam perjalanan ini. Shalawat dan salam selalu tercurah kepada Rasulullah SAW, suri teladan bagi seluruh umat.

Karya ini dipersembahkan kepada ayahanda tercinta, Mohammad Kurdi, sosok yang dengan ketulusan dan pengorbanannya telah menjadi pilar dalam setiap langkah kehidupan ini. Kepada ibunda terkasih, Latifah, yang dalam setiap doanya terselip harapan dan perlindungan, terima kasih atas kasih sayang, kesabaran, dan ketulusan yang tiada batas. Begitu pula kepada kakak tercinta, Silvi Nur Diana, yang senantiasa menjadi teman berbagi dalam suka dan duka, serta penyemangat dalam setiap tantangan.

Selain itu, rasa terima kasih juga disampaikan kepada seseorang yang tanpa aku sadari telah menjadi bagian dari perjalanan ini. Kehadirannya, meski tanpa banyak kata, telah memberi arti dalam setiap langkah yang ditempuh. Tak lupa, untuk diriku sendiri, atas keteguhan dalam menghadapi setiap rintangan, serta atas usaha yang terus diperjuangkan tanpa menyerah. Semoga setiap langkah yang tertuang dalam karya ini membawa kebermanfaatan yang lebih luas dan keberkahan yang tiada terhingga.

KATA PENGANTAR

Puji syukur kehadirat Allah SWT atas limpahan rahmat, nikmat, serta hidayah-Nya sehingga penulis dapat menyelesaikan skripsi yang berjudul "Implementasi Fungsi *Hash* SHA-256 dan Kriptografi Algoritma ECDSA (*Elliptic Curve Digital Signature Algorithm*) dalam Pembuatan Tanda Tangan Digital". Shalawat serta salam senantiasa tercurah kepada Nabi Muhammad SAW, yang telah membawa umat manusia menuju ajaran Islam sebagai pedoman hidup.

Penyusunan skripsi ini tidak terlepas dari bimbingan, dukungan, serta bantuan dari berbagai pihak. Oleh karena itu, dengan penuh rasa hormat dan terima kasih, penulis menyampaikan apresiasi yang sebesar-besarnya kepada:

1. Prof. Dr. H. M. Zainuddin, M.A., selaku Rektor Universitas Islam Negeri Maulana Malik Ibrahim.
2. Prof. Dr. Sri Harini, M.Si., selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim.
3. Dr. Elly Susanti, S.Pd., M.Sc., selaku Ketua Program Studi Matematika, Universitas Islam Negeri Maulana Malik Ibrahim.
4. Muhammad Khudzaifah, M.Si., selaku Dosen Pembimbing I yang telah memberikan arahan serta ilmu yang sangat berharga dalam penyusunan skripsi ini.
5. Erna Herawati, M.Pd., selaku Dosen Pembimbing II yang dengan sabar memberikan bimbingan dan motivasi dalam menyelesaikan penelitian ini.
6. Ari Kusumastuti, M.Pd., M.Si., selaku dosen wali yang telah memberikan arahan dan dukungan selama masa studi.
7. Seluruh dosen Program Studi Matematika, Fakultas Sains dan Teknologi, yang telah memberikan ilmu serta wawasan selama perkuliahan.
8. Kedua orang tua, Mohammad Kurdi dan Latifah, yang dengan penuh cinta, doa, dan pengorbanan tanpa henti selalu memberikan dukungan dalam setiap langkah yang penulis tempuh. Kasih sayang dan doa yang tulus dari mereka menjadi kekuatan terbesar dalam perjalanan akademik ini.

9. Teman-teman mahasiswa angkatan 2021, yang selalu menjadi penyemangat, berbagi pengalaman, serta mendukung dalam perjalanan akademik ini.
10. Seseorang yang tanpa disadari telah menjadi bagian dari perjalanan ini, yang kehadirannya, meski dalam diam, memberikan makna dalam setiap langkah yang ditempuh.

Penulis menyadari bahwa skripsi ini masih memiliki kekurangan. Oleh karena itu, kritik dan saran yang membangun sangat diharapkan untuk perbaikan di masa mendatang. Semoga penelitian ini dapat memberikan manfaat dan kontribusi bagi perkembangan ilmu pengetahuan, khususnya dalam bidang kriptografi.

Malang, 16 Mei 2025

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

ABSTRAK

Millah, Zakiyah Nur. 2025. **Implementasi Fungsi *Hash* SHA-256 dan Kriptografi Algoritma ECDSA (*Elliptic Curve Digital Signature Algorithm*) dalam Pembuatan Tanda Tangan Digital**. Skripsi. Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Muhammad Khudzaifah, M.Si. (II) Erna Herawati, M.Pd.

Kata kunci: ECDSA, keamanan informasi, kurva eliptik, SHA-256, tanda tangan digital.

Tanda tangan digital merupakan metode keamanan dalam sistem informasi yang menjamin autentikasi, integritas, dan non-repudiasi dokumen. Penelitian ini bertujuan untuk mengimplementasikan tanda tangan digital menggunakan fungsi *hash* SHA-256 dan algoritma ECDSA (*Elliptic Curve Digital Signature Algorithm*) dalam sebuah aplikasi berbasis *Python* (PyQt5) melalui empat tahap utama, yaitu penentuan titik pada kurva eliptik, pembangkitan pasangan kunci, proses penandatanganan, dan verifikasi tanda tangan digital. Aplikasi diuji menggunakan 30 dokumen PDF untuk mengukur keandalan dan kinerjanya, dengan hasil menunjukkan bahwa aplikasi berhasil menghasilkan tanda tangan digital dalam bentuk pasangan nilai (r, s) yang sesuai dengan algoritma ECDSA. Pada tahap verifikasi, sistem secara akurat mendeteksi perubahan pada dokumen, tanda tangan digital, atau kunci publik, serta menolak tanda tangan yang tidak valid. Dari 30 dokumen yang diuji, sebanyak 25 dokumen tetap valid, sedangkan lima dokumen lainnya terdeteksi mengalami perubahan sehingga tanda tangannya tidak valid. Keunggulan penelitian ini terletak pada penggunaan algoritma SHA-256 yang lebih aman dibandingkan SHA-1 atau MD5 serta penerapan kurva eliptik dalam bidang hingga yang memperkuat aspek keamanan. Selain itu, penelitian ini mengintegrasikan prinsip keamanan digital dengan konsep amanah dalam Islam, yang menekankan keabsahan dan kepercayaan dalam sistem tanda tangan digital. Dengan demikian, implementasi ini membuktikan bahwa tanda tangan digital berbasis kurva eliptik dapat diterapkan dengan tingkat keamanan yang tinggi dan kinerja yang baik.

ABSTRACT

Millah, Zakiyah Nur. 2025. **Implementation of SHA-256 Hash Function and ECDSA (Elliptic Curve Digital Signature Algorithm) Cryptography in Digital Signature Creation**. Thesis. Department of Mathematics, Faculty of Science and Technology, Maulana Malik Ibrahim State Islamic University of Malang. Advisors: (I) Muhammad Khudzaifah, M.Si. (II) Erna Herawati, M.Pd.

Keywords: digital signatures, ECDSA, elliptic curve, information security, SHA-256.

Digital signatures are a security method in information systems that ensure authentication, integrity, and non-repudation of documents. This study aims to implement digital signatures using the SHA-256 hash function and the ECDSA (Elliptic Curve Digital Signature Algorithm) algorithm in a Python-based application (PyQt5) through four main stages, namely determining the point on the elliptic curve, generating key pairs, signing process, and digital signature verification. The application was tested using 30 PDF documents to measure its reliability and performance, with the results showing that the application successfully generated a digital signature in the form of a value pair (r, s) that conforms to the ECDSA algorithm. At the verification stage, the system accurately detects changes to documents, digital signatures, or public keys, as well as rejects invalid signatures. Out of the 30 documents tested, 25 documents remained valid, while five other documents were detected to have changed so that their signatures were invalid. The advantages of this research lie in the use of the SHA-256 algorithm which is safer than SHA-1 or MD5 and the application of elliptic curves in the field of traffic strengthens the security aspect. In addition, this study integrates the principles of digital security with the concept of trust in islam, which emphasizes validity and trust in the digital signature system. Thus, this implementation proves that elliptic curve-based digital signatures can be implemented with a high level of security and good performance.

مستخلص البحث

الملة، زكية نور. ٢٠٢٥. **تنفيذ وظيفة التجزئة *SHA-256* وخوارزمية تشفير خوارزمية التوقيع الرقميّ *ECDSA* (خوارزمية التوقيع الرقمي للمنحنى البيضاوي) في تصنيع التوقيع الرقميّ**. البحث العلمي. قسم الرياضيات، كلية العلوم والتكنولوجيا، جامعة مولانا مالك إبراهيم الإسلامية الحكومية مالانج. المشرف: (١) محمد حذيفة، الماجستير.(٢) إيرنا هيراواتى، الماجستيرة.

الكلمات المفتاحية: التوقيع الرقمي، *ECDSA*، *SHA-256*، المنحنى البيضاوي، معلومة الأمن.

التوقيع الرقمي هو طريق الأمن في أنظمة المعلومات الذي يضمن توثيق المستندات وسلامته وعدم التنصل منه. هذفت هذه الدراسة إلى تنفيذ التوقيع الرقمي باستخدام وظيفة التجزئة *SHA-256* وخوارزمية *ECDSA* (خوارزمية التوقيع الرقمي للمنحنى البيضاوي) في التطبيق القائم على *Python (PyQt5)* من خلال أربع المراحل الرئيسية، وهو تحديد النقطة على المنحنى البيضاوي، وتوليد أزواج المفاتيح، وعملية التوقيع، والتحقق من التوقيع الرقمي. تم اختبار التطبيق باستخدام ٣٠ وثيق *PDF* لقياس موثوقيته وأدائه، حيث أظهرت النتائج أن التطبيق نجح في تصنيع التوقيع الرقمي على شكل الزوج القيم (r, s) يتوافق مع خوارزمية *ECDSA*. في مرحلة التحقق، يكتشف النظام بدقة التغييرات الذي يطرأ على المستندات أو التوقيع الرقمي أو المفاتيح العامة، بالإضافة إلى رفض التوقيعات غير الصالحة. ومن ٣٠ الوثائق التي اختبارها، مازال إلى ٢٥ وثيقة صالحة، وخمس الوثائق الأخرى غير صحيحة. ومزاية هذا البحث في استخدام خوارزمية *SHA-256* أكثر أمنا من إستخدام خوارزمية *SHA-1* أو *MD5* وتطبيق المنحني البيضاوي في ميدان المرور الذي يعزز الجانب الأمني. إضافة إلى ذلك، دمجت هذه الدراسة مبادئ الأمن الرقمي مع مفهوم الثقة في الإسلام، مما يؤكد على الصلاحية والثقة في نظام التوقيع الرقمي. وبالتالي، يثبت هذا التنفيذ أنه يمكن تنفيذ التوقيع الرقمي القائم على المنحنى البيضاوي بمستوى عال من الأمن والأداء الجيد.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi saat ini telah memengaruhi hampir setiap aspek kehidupan manusia sangat dipengaruhi oleh kemajuan dan perkembangan teknologi, termasuk dalam hal komunikasi. Keberadaan internet telah menghilangkan hambatan jarak, sehingga memungkinkan terjadinya komunikasi yang cepat dan efisien. Internet kini dimanfaatkan untuk berbagai keperluan seperti perdagangan, bisnis, industri, dan pemerintahan. Informasi yang ditransfer melalui internet sering kali memiliki nilai penting atau strategis, mencakup data elektronik yang dihasilkan, dikirim, diterima, dan disimpan dalam berbagai bentuk, baik analog maupun digital. Nilai informasi ini dapat berkaitan dengan aspek ekonomi, kerahasiaan, atau dampaknya terhadap kelancaran operasional bisnis dan pemerintahan.

Dalam penggunaannya, data elektronik memiliki kelemahan dalam keotentikannya, karena sangat mudah diubah dan diduplikasikan. Hal ini membuka kemungkinan penyalahgunaan oleh individu yang tidak berwenang. Akibatnya, penting untuk memastikan bahwa data yang dikirimkan sampai kepada penerima yang berhak. Salah satu solusi untuk masalah ini adalah kriptografi kunci publik, yang sangat berguna dalam penerapan tanda tangan digital. Tanda tangan digital berfungsi sebagai sarana autentikasi dokumen, mirip dengan tanda tangan manual. Setiap dokumen yang menggunakan tanda tangan digital memiliki identitas unik, sehingga pengirim dapat mengaitkan dokumen dengan kode autentikasi yang sah.

Tanda tangan digital ini dibuat menggunakan variasi *Elliptic Curve Cryptography* (ECC) yang dikenal sebagai *Elliptic Curve Digital Signature Algorithm* (ECDSA), sebuah metode kriptografi yang mempermudah proses autentikasi. Dibandingkan dengan metode kriptografi kunci publik lainnya, ECDSA menawarkan keamanan yang tinggi dengan panjang kunci yang lebih pendek. Selain itu, ECDSA memanfaatkan konsep matematika kurva eliptik untuk menjamin integritas dan autentikasi data, sekaligus mempertahankan tingkat efektivitas dan kinerja optimal selama proses pertukaran kunci. Keunggulan utama ECDSA dibandingkan algoritma tanda tangan digital lainnya adalah menerapkan tingkat keamanan yang sama pada panjang kunci yang lebih pendek (Mallouli et al., 2019). Algoritma ECDSA semakin banyak digunakan dalam aplikasi tanda tangan digital karena efisiensi dan keandalannya dalam menjaga keamanan komunikasi digital.

Konsep keamanan dan autentikasi data melalui teknologi kriptografi dapat dikaitkan dengan prinsip amanah yang diajarkan dalam islam. Dalam Surah Al-Mu'minun ayat 8, Allah SWT menekankan pentingnya menjaga amanah dan menunaikannya dengan penuh tanggung jawab: *"Dan orang-orang yang memelihara amanat-amanat (yang dipikulnya) dan janjinya"* (QS. Al-Mu'minun: 8). Prinsip ini sejalan dengan upaya memastikan bahwa data dan informasi yang dipercayakan tetap terjaga keamanannya dan di sampaikan kepada pihak yang berhak secara benar dan adil. Amanah dalam ayat ini bukan hanya berlaku dalam hubungan antar manusia, tetapi juga dalam konteks modern, termasuk pengelolaan data digital. Dalam dunia digital, data yang memiliki nilai strategis dan ekonomis harus dijaga keasliannya dan dikirimkan hanya kepada yang berhak. Kriptografi,

khususnya melalui tanda tangan digital berbasis ECDSA, berfungsi untuk memastikan bahwa amanah ini terpenuhi. Amanah dalam hal ini berarti menjaga integritas dan keotentikan data agar tidak disalahgunakan oleh pihak yang tidak berwenang.

Tanda tangan digital dengan algoritma ECDSA menjadi salah satu wujud nyata dari penerapan amanah dalam era teknologi informasi. Setiap data yang dikirimkan melalui internet dengan tanda tangan digital diikat oleh tanggung jawab moral dan spiritual untuk memastikan bahwa data tersebut sampai kepada penerima yang benar dan tetap terjaga keutuhannya. Selain itu, algoritma ini menjaga agar proses autentikasi berjalan adil dan efisien, tanpa ada pihak yang dirugikan.

Dalam penelitian sebelumnya yang dilakukan oleh Nabila Arwa, Aminuddin, dan Sofyan Arifianto, telah diterapkan algoritma ECDSA dengan fungsi *hash* SHA-1 untuk pembuatan tanda tangan digital pada dokumen PDF melalui *platform web*. Studi ini menitikberatkan pada penerapan ECDSA dan evaluasi aplikasinya dalam lingkungan *web* dengan jumlah uji sebanyak sepuluh dokumen PDF. Hasil tanda tangan ditampilkan dalam bentuk heksadesimal, dan aplikasi diuji untuk mengamati kelayakan serta kemampuannya dalam menandatangani dokumen PDF secara digital. Namun, penelitian ini memiliki keterbatasan pada penggunaan fungsi *hash* SHA-1, yang saat ini kurang disarankan karena kelemahan dalam aspek keamanan akibat potensi kerentanannya terhadap serangan *collision.*

Sementara itu, pada penelitian yang dilakukan oleh Annisa Hardiningsih HR menggunakan fungsi *hash* MD5 dan algoritma RSA untuk membuat tanda tangan digital pada dokumen PDF. Penelitian ini terfokus pada pembuatan tanda tangan digital untuk dokumen yang berisikan alfabet, angka, dan karakter khusus tertentu

(~`@#$%^&*()-_+=[]{}|:;’”<>,.?/). Hasil penelitian ini merekomendasikan agar studi lanjutan mencakup dokumen PDF yang tidak hanya berisikan karakter teks, namun juga mendukung elemen gambar atau tabel untuk meningkatkan fleksibilitas dan kegunaan tanda tangan digital.

Penelitian sejenis yang dilakukan oleh Alfie Amanilla Aziz, membahas penggunaan metode Ong-Schnorr-Shamir dan algoritma Euclidean untuk menghasilkan tanda tangan digital pada data berbentuk teks. Fokus penelitian ini adalah penerapan metode kriptografi klasik dalam pembuatan tanda tangan digital dan evaluasi ketepatan serta efisiensi metode dalam konteks keamanan pada teks murni. Namun, penelitian ini memiliki keterbatasan karena hanya mencakup teks tanpa elemen gambar atau tabel. Selain itu, metode Ong-Schnorr-Shamir memiliki kompleksitas yang tinggi, sehingga kurang efisien untuk skala besar atau data yang lebih kompleks. Penggunaan metode ini juga kurang optimal dalam konteks keamanan modern yang membutuhkan tingkat keamanan lebih tinggi.

Studi serupa yang dilakukan oleh Anwar Ramadha, mengusulkan penggabungan antara *one-time password* (OTP) dan algoritma kriptografi kunci publik untuk otentikasi yang lebih aman. Penelitian ini memanfaatkan kriptografi kurva eliptik (*Elliptic Curve*) untuk validasi OTP, sehingga meskipun terjadi serangan seperti *man-in-the-middle*, OTP yang dihasilkan tidak dapat disalahgunakan tanpa akses ke kunci privat. Walaupun metode ini terbukti efektif dalam menjaga keamanan otentikasi, penelitian ini masih memiliki kekurangan, yaitu belum adanya mekanisme untuk mengatur masa kedaluwarsa OTP yang tidak valid setelah beberapa kali percobaan gagal. Penulis merekomendasikan pengembangan sistem yang dapat mengidentifikasi dan membuat OTP kadaluwarsa

jika terjadi percobaan berulang yang gagal, guna meningkatkan keamanan lebih lanjut.

Penulis tertarik untuk menyelidiki penggunaan algoritma kriptografi ECDSA dan fungsi *hash* SHA-256 untuk implementasi tanda tangan digital, sebagaimana telah dijelaskan pada penelitian-penelitian sebelumnya. Penelitian ini diharapkan tidak hanya menginspirasi penelitian lebih lanjut tentang tanda tangan digital, tetapi juga memberikan kontribusi signifikan bagi perkembangan teknologi keamanan informasi. Selain itu, penelitian ini bertujuan untuk mengintegrasikan konsep amanah dalam Islam ke dalam dunia digital, sebagai upaya menjaga integritas dan keadilan dalam setiap proses pertukaran informasi.

Dalam penelitian ini, pendekatan yang sejenis digunakan, namun dengan penekanan tambahan pada aspek matematis yang lebih mendetail serta pemanfaatan fungsi *hash* yang lebih kuat, yaitu SHA-256. Implementasi dilakukan melalui aplikasi desktop berbasis *Python* yaitu PyQt5. Untuk mendukung keutuhan analisis, tiga puluh dokumen PDF diuji dalam aplikasi ini. Hal ini memungkinkan penelitian untuk mengevaluasi keandalan tanda tangan digital pada beragam jenis konten dalam dokumen PDF. Selain itu, dilakukan perhitungan manual yang memungkinkan perbandingan jelas antara hasil dari aplikasi dan hasil matematis, sehingga proses dapat dipahami lebih komprehensif.

Meskipun nilai-nilai yang digunakan dalam aplikasi ini bersifat sederhana, hal ini bertujuan untuk memudahkan pemahaman proses. Penelitian ini tidak hanya menunjukkan keandalan ECDSA dan SHA-256 sebagai metode autentikasi yang aman, tetapi juga berfungsi sebagai sarana pembelajaran yang mendemonstrasikan secara rinci proses matematis dalam pembuatan tanda tangan digital. Hasil dari

proses tersebut ditampilkan dalam bentuk pasangan titik (r, s), agar setiap tahap matematis yang dilalui dapat dipahami dengan lebih struktural.

Penelitian-penelitian sebelumnya telah menerapkan algoritma ECDSA untuk tanda tangan digital pada dokumen PDF yang mencakup teks, namun kurang memperhatikan penggunaan fungsi *hash* yang lebih kuat seperti SHA-256, yang memiliki tingkat keamanan lebih tinggi dibandingkan SHA-1 atau MD5. Selain itu, aspek matematis belum dijelaskan secara terperinci untuk mendukung pemahaman mendalam. Oleh karena itu, penelitian ini berkontribusi dengan mengembangkan pendekatan yang lebih aman dan aplikatif pada lingkungan desktop yang lebih fleksibel.

1.2 Rumusan Masalah

Permasalahan pada penelitian ini dirumuskan sebagai berikut:

1. Bagaimana hasil implementasi tanda tangan digital menggunakan fungsi *hash* SHA-256 dan kriptografi algoritma ECDSA?
2. Bagaimana hasil verifikasi tanda tangan digital menggunakan fungsi *hash* SHA-256 dan kriptografi algoritma ECDSA?

1.3 Tujuan Penelitian

Berdasarkan permasalahan yang telah diuraikan di atas, maka tujuan penelitian ini adalah:

1. Mengimplementasikan tanda tangan digital menggunakan fungsi *hash* SHA-256 dan kriptografi algoritma ECDSA.
2. Melakukan verifikasi tanda tangan digital menggunakan fungsi *hash* SHA-256 dan kriptografi algoritma ECDSA.

1.4 Manfaat Penelitian

Manfaat dari penelitian ini antara lain:

1. Meningkatkan keamanan data digital melalui penerapan algoritma ECDSA dan fungsi *hash* SHA-256 yang efektif dalam menjaga integritas dan otentisitas data, sehingga data lebih aman dari manipulasi maupun akses ilegal.
2. Menghasilkan aplikasi tanda tangan digital yang efisien untuk penandatanganan dan verifikasi tanda tangan digital.
3. Menjadi referensi akademis bagi akademisi dan praktisi dalam implementasi ECDSA dan SHA-256 pada tanda tangan digital.
4. Berfungsi sebagai sumber pembelajaran bagi mahasiswa dengan analisis matematis mendalam tentang algoritma yang digunakan.
5. Mendorong pengembangan lebih lanjut dalam teknologi tanda tangan digital dan meningkatkan kesadaran akan pentingnya kriptografi yang kuat.

1.5 Batasan Masalah

Permasalahan yang menjadi fokus dalam penelitian ini adalah sebagai berikut:

1. Fokus implementasi terletak pada aplikasi dekstop menggunakan *Python* dan PyQt5.
2. Analisis matematis terbatas pada pasangan titik (r, s) tanpa membahas angka yang lebih besar.
3. Dokumen yang digunakan berisi teks sebagai bahan uji aplikasi.

1.6 Definisi Istilah

Kriptografi : Cabang ilmu yang mempelajari teknik matematika dalam keamanan informasi, meliputi verifikasi identitas (autentikasi), integritas data, serta kerahasiaan.

Tanda Tangan Digital : Sebuah sistem autentikasi yang memungkinkan pengirim pesan untuk menyertakan sebuah kode yang berperan sebagai tanda tangannya.

Plaintext : Informasi asli sebelum di enkripsi.

Enkripsi : Suatu transfromasi dari *plaintext* menjadi *chipertext*.

Chipertext : Data acak yang berasal dari *plaintext* yang telah diproses secara matematis menggunakan fungsi dan algoritma.

Dekripsi : Suatu transformasi dari *chipertext* menjadi *plaintext*.

Kriptoanalisis : Proses mengubah *chipertext* menjadi *plaintext* dengan tetap menjaga kerahasiaan kuncinya.

Kriptoanalis : Seseorang yang terlibat dalam kriptoanalisis

Kriptologi : Pengetahuan mengenai kriptografi dan kriptoanalisis.

Sistem Kriptografi : Sebuah himpunan yang mencakup algoritma kriptografi, semua kemungkinan *plaintext* dan *chipertext*, serta kunci yang terlibat.

Penyadap : Seseorang yang berupaya untuk mengetahui pesan ketika pesan dikirim atau di tranmisikan.

BAB II

KAJIAN TEORI

2.1 Teori Aljabar

2.1.1 Grup

Menurut Joseph A. Gallian, suatu himpunan $\mathbb{G}$ dikatakan sebagai grup jika dilengkapi dengan operasi biner (biasanya disebut perkalian) yang memasangkan setiap elemen a dan b dalam $\mathbb{G}$ dengan elemen lain dalam $\mathbb{G}$, yang dilambangkan sebagai ab. Agar $\mathbb{G}$ membentuk grup, maka harus memenuhi aksioma-aksioma berikut (Gallian, 2021):

1. Asosiatif: Operasi bersifat asosiatif, untuk setiap $a, b, c \in \mathbb{G}$, berlaku $(ab)c = a(bc)$.
2. Elemen identitas : Terdapat $e \in \mathbb{G}$ sedemikian rupa sehingga $ae = ea = a$ untuk setiap $a \in \mathbb{G}$.
3. Elemen Invers : setiap elemen $a \in \mathbb{G}$ memiliki elemen $b \in \mathbb{G}$ (disebut invers a) sehingga $ab = ba = e$, dimana e adalah elemen identitas.

2.1.2 Ring

Suatu himpunan tak kosong R disebut ring jika memiliki dua operasi tertutup, yaitu penjumlahan $(+)$ dan perkalian, yang memenuhi aksioma-aksioma berikut (Judson, 2020):

1. Penjumlahan bersifat komutatif: $a + b = b + a$ untuk semua $a, b \in R$.
2. Penjumlahan bersifat asosiatif: $(a + b) + c = a + (b + c)$ untuk semua $a, b, c \in R$.
3. Terdapat elemen nol 0 dalam R sebagai identitas penjumlahan, sehingga

$a + 0 = a$ untuk semua $a \in R$.

4. Setiap elemen $a \in R$ memiliki elemen balikan $-a$ dalam R, sehingga

   $a + (-a) = 0$.

5. Perkalian bersifat asosiatif: $(ab)c = a(bc)$ untuk setiap $a, b, c \in R$.

6. Aksioma distributif mengaitkan kedua operasi, yaitu:

   $a(b + c) = (ab) + (ac)$.

   $(a + b)c = ac + bc$ untuk semua $a, b, c \in R$.

Empat syarat pertama menjelaskan bahwa R merupakan grup abelian dengan operasi penjumlahan. Jadi, sebuah ring dapat dianggap sebagai grup abelian di bawah penjumlahan dengan operasi perkalian tambahan yang memenuhi aksioma kelima dan keenam.

2.1.3 *Field*

Medan, atau *field*, merupakan struktur aljabar yang beroperasi sebagai ring komutatif, di mana setiap elemen memiliki kebalikan perkalian, kecuali elemen nol. Dalam medan F, untuk setiap elemen $a \neq 0$, terdapat elemen a^{-1}, yang memenuhi persamaan $a(a^{-1}) = 1$. Medan dapat terbentuk dari himpunan bilangan real atau rasional, yang mendukung operasi penjumlahan dan perkalian. Medan berhingga, yang memiliki jumlah elemen terbatas, sering dimanfaatkan dalam kriptografi, sedangkan medan tak berhingga seperti bilangan real umumnya tidak diigunakan dalam aplikasi ini karena sifat operasional.

2.1.4 *Elliptic Curve* Membentuk Sebuah Grup

Kurva eliptik membentuk grup $\mathbb{G}$ dengan operasi penjumlahan yang dinyatakan sebagai ($\mathbb{G}$, +). Himpunan $\mathbb{G}$ terdiri dari semua titik $P(x, y)$ yang

terletak pada kurva eliptik tersebut, sementara operasi biner yang digunakan adalah penjumlahan (+). Operasi biner ini memiliki peran penting dalam berbagai aplikasi matematika, termasuk di bidang kriptografi, di mana struktur grup dari kurva eliptik digunakan untuk memastikan keamanan dalam sistem enkripsi dan tanda tangan digital.

Jika P dan Q merupakan titik-titik dalam himpunan $\mathbb{G}$, seluruh aksioma yang diperlukan untuk membentuk grup telah dipenuhi, sebagaimana dijelaskan dalam subbab sebelumnya. Dengan demikian, $\mathbb{G}$ membentuk grup di bawah operasi penjumlahan titik-titik tersebut.

2.1.5 *Elliptic Curve* Pada *Galois Field*

Bentuk umum *elliptic curve* pada $GF(p)$ (atau Fp) adalah:

$$y^2 \equiv x^3 + ax + b \ (mod\ p)$$

Elemen-elemen dalam medan galois dalam hal ini adalah $\{0, 1, 2, \ldots, p-1\}$ dimana p adalah bilangan prima. Menurut Munir (2019) modulus p digunakan untuk semua operasi penjumlahan dan perkalian.

Contoh:

Pada kurva eliptik $y^2 \equiv x^3 + x + 6 \ (mod\ 11)$, tentukan semua titik $P(x, y)$ dengan x dan y di definisikan di dalam $GF(11)$.

Penyelesaian:

Elemen-elemen di $GF(11)$ yaitu $\{0, 1, 2, \ldots, 10\}$. Kemudian hitung semua titik yang terletak pada kurva eliptik:

$x = 0 \ \rightarrow \ y^2 \equiv 6 \ (mod\ 11)$

$x = 1 \ \rightarrow \ y^2 \equiv 8 \ (mod\ 11)$

$x = 2 \ \rightarrow y^2 \equiv 16 \ mod\ 11 \equiv 5 \ (mod\ 11) \rightarrow y_1 = 4 \ dan \ y_2 = 7 \rightarrow P_1(2, 4)$

dan $P_2(2,7)$

$x = 3 \rightarrow y^2 \equiv 36\ mod\ 11 \equiv 3\ (mod\ 11) \rightarrow y_1 = 5\ dan\ y_2 = 6 \rightarrow P_1(3,5)$

dan $P_2(3,6)$

⋮

$x = 10 \rightarrow y^2 \equiv 1.016\ mod\ 11 \equiv 4(mod\ 11) \rightarrow y_1 = 2\ dan\ y_2 = 9 \rightarrow$

$P_1(10,2)$ dan $P_2(10,9)$

Jadi, kurva eliptik $y^2 \equiv x^3 + x + 6\ (mod\ 11)$ memiliki 12 titik, yaitu: $(2,4), (2,7), (3,5), (3,6), (5,2), (5,9), (7,2), (7,9), (8,3), (8,8), (10,2), (10,9)$. Titik-titik pada kurva eliptik membentuk grup dengan $n = 13$ anggota jika dihubungkan dengan titik O di *infinity*.

2.2 Kriptografi

Istilah "kriptografi" berasal dari dua kata dalam bahasa Yunani, yaitu "*cryptos*" yang berarti "rahasia" atau "tersembunyi" dan "*graphein*" yang berarti "tulisan". Secara keseluruhan, kriptografi dapat dipahami sebagai "tulisan tersembunyi" atau "tulisan rahasia". Berbagai definisi kriptografi dijumpai dalam literatur. Sebagai contoh, kriptografi diartikan sebagai ilmu dan seni untuk menjaga kerahasiaan pesan, serta studi tentang teknik matematika untuk menangani masalah keamanan informasi, termasuk integritas data dan verifikasi autentikasi.

Keamanan merupakan konsep yang integral dalam kriptografi, mencakup pengelolaan kerahasiaan informasi, autentikasi integritas data, dan memastikan bahwa pengirim pesan adalah pihak yang sah, serta mencegah pengirim menyangkal pengiriman pesan tersebut.

Dalam konteks algoritma kriptografi, terdapat tiga fungsi utama yang berperan penting. Pertama, enkripsi, yang merupakan proses pengacakan pesan

menggunakan kunci untuk menjaga kerahasiaan isi pesan dengan mengubah teks biasa menjadi bentuk yang tidak terbaca. Kedua, dekripsi, yang berfungsi mengembalikan pesan terenkripsi ke bentuk aslinya dengan menggunakan algoritma yang berbeda dari enkripsi. Ketiga, kunci, yang meliputi kunci publik dan kunci pribadi yang digunakan dalam proses enkripsi dan dekripsi (Munir, 2019).

2.2.1 Terminologi Dalam Kriptografi

Beberapa terminologi yang sering digunakan dalam bidang kriptografi mencakup: pesan, pengirim, penerima, *chiper*, kunci, kode, enkripsi, dekripsi, penyadap, plainteks, chiperteks, kriptoanalisis, kriptologi, dan sistem kriptografi.

1. Pesan, Plainteks, dan Chiperteks

   Pesan merupakan segala bentuk informasi yang dapat diuraikan dan dipahami, dan dapat dikirim dalam format digital maupun analog, termasuk teks, gambar, video, audio, atau format biner lainnya. Komunikasi teks sering disebut sebagai plainteks, sedangkan komunikasi dalam bentuk gambar, video, dan audio dikenal sebagai *plain-image, plain-video, dan plain-audio*. Agar isi komunikasi tidak dapat dipahami oleh pihak ketiga, pesan harus dienkripsi. Chiperteks mengacu pada teks yang telah dienkripsi, sedangkan *chiper-image, chiper-video, dan chiper-audio* merujuk pada gambar, video, dan audio yang telah terenkripsi. Pesan yang telah dienkripsi dapat didekripsi untuk dikembalikan ke bentuk yang dapat dipahami (plainteks).

2. Enkripsi dan Dekripsi

   Proses yang mengubah plainteks menjadi chiperteks menggunakan kunci dikenal sebagai enkripsi. Sebaliknya, proses mengembalikan

chiperteks ke plainteks disebut dekripsi. Dalam enkripsi, plainteks dan kunci digunakan sebagai input untuk menghasilkan chiperteks sebagai *output*. Di sisi lain, dekripsi mengambil chiperteks dan kunci sebagai input untuk mengembalikan pesan ke bentuk aslinya. Metode ini mengubah pesan yang awalnya dapat dimengerti menjadi pesan yang tidak dapat dipahami, sehingga isi pesan diacak sedemikian rupa.

3. Pengirim dan Penerima

   Dalam pertukaran komunikasi, terdapat dua pihak utama, yaitu pengirim dan penerima. Pengirim adalah entitas yang menyampaikan pesan, sedangkan penerima adalah entitas yang menerima pesan tersebut. Pengirim tidak hanya terbatas pada individu, tetapi juga dapat berupa komputer, mesin, atau robot (Munir, 2019).

4. Sistem Kriptografi

   Sistem yang digunakan dalam kriptografi disebut *cryptosystem*, yang mencakup kumpulan plainteks, chiperteks, ruang kunci, serta teknik enkripsi dan dekripsi. Terdapat dua kategori sistem kriptografi: sistem kriptografi kunci-simetris dan kunci-publik. Prosedur enkripsi dan dekripsi dalam kedua sistem ini berbeda; pada kunci-simetris, kunci yang sama digunakan untuk kedua proses, sementara pada kunci-publik, kunci yang berbeda digunakan.

5. Kriptanalisis dan Kriptologi

   Kriptanalisis adalah studi tentang pemecahan chiperteks tanpa mengetahui kunci yang digunakan dalam enkripsi dan dekripsi. Individu yang melakukan kriptanalisis disebut kriptanalis, yang berupaya

menguraikan chiperteks untuk mengungkap plainteks atau kunci. Kriptologi mencakup kedua bidang ini, berfokus pada studi kriptografi dan teknik kriptoanalisis.

6. Penyadap

   Penyadap adalah individu yang menyadap komunikasi untuk memperoleh informasi yang dibicarakan selama transmisi. Penyadap berusaha mengumpulkan sebanyak mungkin informasi untuk memahami teknologi enkripsi yang digunakan, dengan tujuan untuk berhasil menguraikan chiperteks.

7. Chiper, Kode, dan Kunci

   Chiper merujuk pada algoritma kriptografi yang digunakan dalam proses enkripsi dan dekripsi, diartikan sebagai serangkaian aturan yang mengatur kedua proses tersebut. Dalam kriptografi modern, penggunaan kunci umumnya diperlukan untuk menjaga keamanan algoritma, di mana kunci merupakan elemen data yang menentukan cara kerja metode kriptografi.

2.2.2 Layanan Kriptografi

Terdapat empat layanan kriptografi yang signifikan, diantaranya (Schneier, 1996):

1. Kerahasiaan

   Tujuan dari kerahasiaan data adalah memastikan bahwa informasi hanya dapat diakses oleh pihak yang berwenang, sehingga pihak yang tidak berwenang tidak memiliki akses terhadapnya. Kerahasiaan informasi dapat dipertahankan melalui berbagai teknik, mulai dari algoritma matematika

yang mengacak data hingga tindakan pengamanan fisik, seperti penyimpanan data di lokasi yang aman.

2. Integritas Data

Prinsip integritas data bertujuan untuk melindungi informasi dari penambahan, penghapusan, atau modifikasi yang tidak sah oleh pihak yang tidak berwenang. Integritas data mengonfirmasi bahwa pesan yang diterima adalah asli dan tidak mengalami perubahan. Sistem keamanan harus dapat mendeteksi manipulasi pesan, seperti penggantian, modifikasi, atau penghapusan oleh pihak yang tidak berwenang. Tanda tangan digital berfungsi untuk mencegah pelanggaran integritas data, dengan memverifikasi bahwa pesan tersebut asli dan belum diubah.

3. Autentikasi

Autentikasi berkaitan dengan proses identifikasi untuk memverifikasi kebenaran informasi yang disampaikan dan mengidentifikasi pelaku. Dalam komunikasi, semua pihak harus saling mengautentikasi untuk memastikan validitasnya. Rincian seperti tanggal, isi data, dan waktu pengiriman harus dipastikan. Dalam kriptografi, autentikasi terbagi menjadi dua bagian: verifikasi keaslian data dan identifikasi pelaku kejahatan. Autentikasi juga menjaga integritas data dengan memastikan bahwa komunikasi diarahkan kepada penerima yang tepat dan berasal dari pengirim yang terverifikasi. Tanda tangan digital digunakan sebagai alat autentikasi.

4. Anti-penyangkalan

Tujuan anti-penyangkalan adalah mencegah pihak yang terlibat dalam komunikasi menyangkal tindakan yang telah dilakukan. Dengan demikian,

pengirim pesan tidak dapat menyangkal bahwa mereka telah mengirim pesan tersebut, dan penerima tidak dapat menyangkal bahwa mereka telah menerimanya. Dalam konteks kriptografi, tanda tangan digital merupakan implementasi dari prinsip anti-penyangkalan, memberikan bukti autentik yang kuat mengenai identitas pengirim dan keabsahan penerima pesan.

2.3 Kriptografi Modern

Kriptografi modern berbeda dari kriptografi tradisional karena melibatkan penggunaan komputasi atau program. Enkripsi modern memanfaatkan komputasi untuk melindungi data yang ditransfer melalui jaringan komputer. Oleh karena itu, memastikan keamanan, integritas, dan keaslian data menjadi sangat penting.

Dalam kriptografi kontemporer, elemen seperti plainteks, kunci, dan chiperteks sering dikonversi menjadi rangkaian digit biner yang terdiri dari angka 0 dan 1. Teknik pengkodean yang umum digunakan adalah ASCII (*American Standard Code for Information Interchange*).

Kunci simetris dan asimetris merupakan dua jenis kunci yang digunakan dalam kriptografi modern. Kunci simetris dibagi menjadi dua kategori, yaitu *block cipher* dan *stream cipher*. Pada *block cipher*, data dibagi menjadi blok-blok berukuran tetap dan dienkripsi secara terpisah untuk setiap blok. Sebaliknya, *stream cipher* menggunakan urutan bit tertentu untuk mengenkripsi data secara bit demi bit.

Berikut adalah beberapa contoh cara merepresentasikan bilangan bulat dalam bentuk biner (basis 2), heksadesimal (basis 16), dan desimal (basis 10). Bilangan bulat desimal dari 0 hingga 15 dapat diwakili dalam 4 bit sebagai berikut:

Tabel 2.1 Representasi Desimal, Heksadesimal, dan Bit

Dec	Hexa	Bit
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000

Dec	Hexa	Bit
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Berbagai jenis algoritma dalam kriptografi modern telah mengalami perkembangan untuk memenuhi berbagai kebutuhan keamanan data. Secara umum, algoritma ini dapat diklarifikasikan ke dalam tiga kategori utama, yaitu algoritma simetris, algoritma asimetris, dan kombinasi dari keduanya.

1. Algoritma Simetris (Kunci Privat)

   Algoritma simetris merupakan jenis algoritma kriptografi di mana kunci yang digunakan untuk enkripsi adalah sama dengan kunci yang digunakan untuk dekripsi. Oleh karena itu, algoritma ini sering disebut sebagai algoritma kunci rahasia atau *single-key algorithm*. Beberapa algoritma yang menggunakan pendekatan ini antara lain:

   a) DES (*Data Encryption Standart*)

   b) AES (*Advance Encryption Standart*)

   c) IDEA (*International Data Encryption Algorithm*)

   d) A5

   e) RC4

   Adapun keunggulan algoritma simetris meliputi:

   a) Proses operasinya lebih cepat dibandingkan algoritma asimetris,

dirancang untuk menyediakan enkripsi dan dekripsi yang cepat.

b) Ukuran kunci yang relatif lebih kecil.

c) Algoritma ini berfungsi dengan baik dalam kondisi real-time karena memiliki kecepatan tinggi.

Sedangkan kelemahan dari algoritma simetris mencakup:

a) Setiap pengguna memerlukan kunci yang unik untuk setiap pertukaran pesan, yang dapat menyulitkan manajemen kunci.

b) Jumlah kunci meningkat secara eksponensial seiring bertambahnya jumlah pengguna, dihitung dengan rumus $n(n-1)/2$

c) Pengiriman kunci harus dilakukan melalui saluran yang aman, sehingga kedua pihak yang terlibat dalam komunikasi harus memastikan kerahasiaan kunci tersebut. Permasalahan ini dikenal sebagai "*key distribution problem*".

d) Kunci perlu diperbarui secara berkala, mungkin setiap kali sesi komunikasi berlangsung.

2. Algoritma Asimetris (Kunci Publik)

Algoritma asimetris memanfaatkan dua kunci kriptografi, di mana satu kunci digunakan untuk proses enkripsi dan kunci lainnya untuk proses dekripsi. Setiap individu dapat mengenkripsi pesan menggunakan kunci publik, tetapi hanya pemilik kunci pribadi yang berwenang untuk mendekripsi pesan tersebut. Beberapa algoritma yang menerapkan konsep asimetris ini meliputi:

a) RSA (*Rivest-Shamir-Adleman*)

b) DSA (*Digital Signature logartihm*)

c) ECC (*Elliptic Curve Cryptiography*)

d) DH (Diffie-Hellman)

Adapun kelebihan algoritma asimetrik antara lain:

a) Hanya kunci pribadi yang harus dijaga kerahasiaannya sepenuhnya.

b) Perubahan pada kunci publik dan kunci pribadi jarang terjadi.

c) Masalah keamanan dapat diatasi dengan lebih baik melalui distribusi kunci yang lebih efisien.

d) Pengelolaan kunci menjadi lebih efektif karena jumlah kunci yang lebih sedikit.

e) Dibandingkan dengan skema kunci simetris, algoritma ini memerlukan jumlah kunci yang lebih sedikit.

Sedangkan kelemahan dari algoritma asimetris mencakup:

a) Proses enkripsi dan dekripsi berlangsung lebih lambat dibandingkan algoritma simetris, mengingat algoritma ini memerlukan komputasi yang lebih intensif.

b) Kunci yang digunakan harus lebih panjang dibandingkan metode simetris untuk mencapai tingkat keamanan yang setara, karena ukuran kunci yang lebih besar diperlukan.

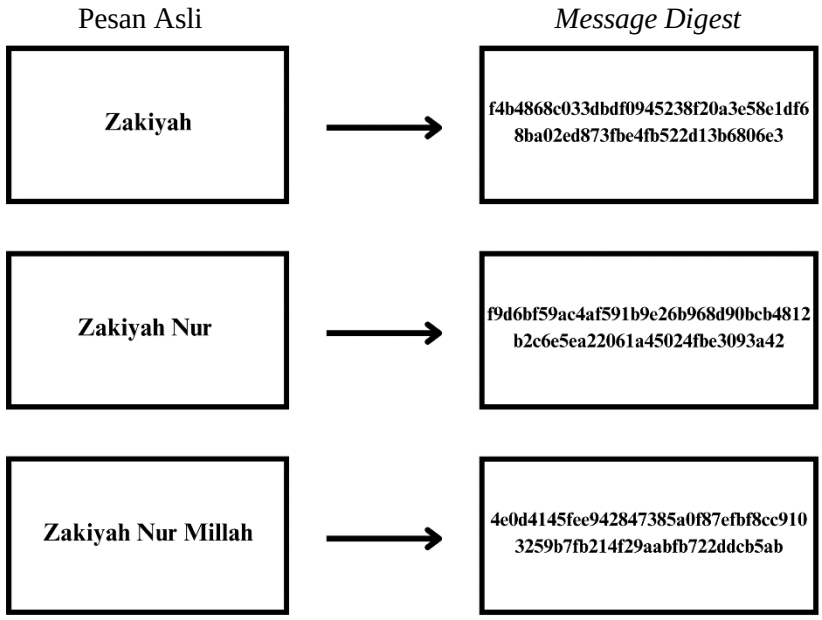
c) Keamanan kunci publik tidak sepenuhnya terjamin.

3. Fungsi *Hash*

Fungsi *hash* merupakan representasi matematis yang mengubah input dengan panjang variabel menjadi urutan biner dengan panjang tetap. Fungsi ini juga dikenal sebagai fungsi *hash* satu arah, intisari pesan, sidik jari, fungsi kompresi, dan kode autentikasi pesan (*Message Authentication Code*

atau MAC).

Fungsi hash sering digunakan untuk menghasilkan sidik jari pesan, yang berperan penting dalam memastikan keabsahan komunikasi serta menandakan bahwa pesan tersebut berasal dari sumber yang terpercaya dan tidak mengalami perubahan selama proses transmisi. Gambar 2.1 berikut menunjukkan bahwa fungsi *hash* menghasilkan *message digest* dengan panjang tetap, terlepas dari variasi ukuran teks input. Konsistensi output ini sangat penting untuk memastikan integritas data, di mana setiap pesan, meskipun berbeda panjang, dilipat menjadi kunci unik yang berukuran sama.



Gambar 2.1 Penggunaan Fungsi *Hash*

Setelah diolah, pesan asli diubah menjadi *message digest* yang bersifat satu arah, sehingga tidak mungkin dikembalikan ke bentuk aslinya. Hal ini mendukung berbagai metode bagi pengirim dan penerima untuk memverifikasi bahwa pesan tidak mengalami perubahan selama transmisi. Untuk memahami bagaimana fungsi *hash* beroperasi, penting untuk

mengetahui parameter-parameter yang terlibat dalam proses ini. Tabel 2.2 di bawah ini menyajikan parameter-parameter umum dari fungsi *hash* beserta keterangan masing-masing.

Tabel 2.2 Parameter Fungsi *Hash*

Parameter	**Keterangan**
M	Pesan
h	Fungsi *Hash*
y	Merupakan *h(M)* atau *message digest*

Dengan memahami parameter-parameter ini, kita dapat lebih mudah mengapresiasi bagaimana fungsi *hash* bekerja dan relevansinya dalam menjaga integritas data. Hal ini menjadi fondasi penting dalam aplikasi kriptografi yang memerlukan tingkat keamanan tinggi.

2.4 Tanda Tangan Digital

Tanda tangan digital adalah bentuk kriptografi modern yang dihasilkan melalui alat elektronik, seperti pena digital atau pemindai, yang berfungsi untuk memastikan keabsahan dan keamanan data dalam dokumen elektronik. Tanda tangan ini merupakan nilai kriptografi yang unik, bergantung pada isi pesan dan identitas pengirim, sehingga setiap pesan yang dikirimkan menghasilkan tanda tangan digital yang berbeda.

Prosesnya melibatkan metode *hashing* yang menciptakan *message digest* sebuah sidik jari digital yang akan berubah jika terdapat modifikasi pada data, sekecil apa pun. Penerima dapat memverifikasi dokumen yang diterima dengan membandingkan *message digest* pengirim dan penerima, sehingga terjamin bahwa dokumen tersebut tidak mengalami perubahan.

Tanda tangan digital memiliki banyak aplikasi dalam menjamin keabsahan dokumen elektronik dan mencegah pemalsuan atau penyalahgunaan informasi. Undang-Undang Informasi dan Transaksi Elektronik (UU ITE) Pasal 11 dan 13 juga memberikan dasar hukum yang kuat bagi tanda tangan elektronik di Indonesia, termasuk tanda tangan digital, untuk menegakkan integritas dan keaslian data digital.

Penandatanganan pesan dapat dilakukan dengan dua metode utama, yaitu (Munir, 2019):

1. Enkripsi

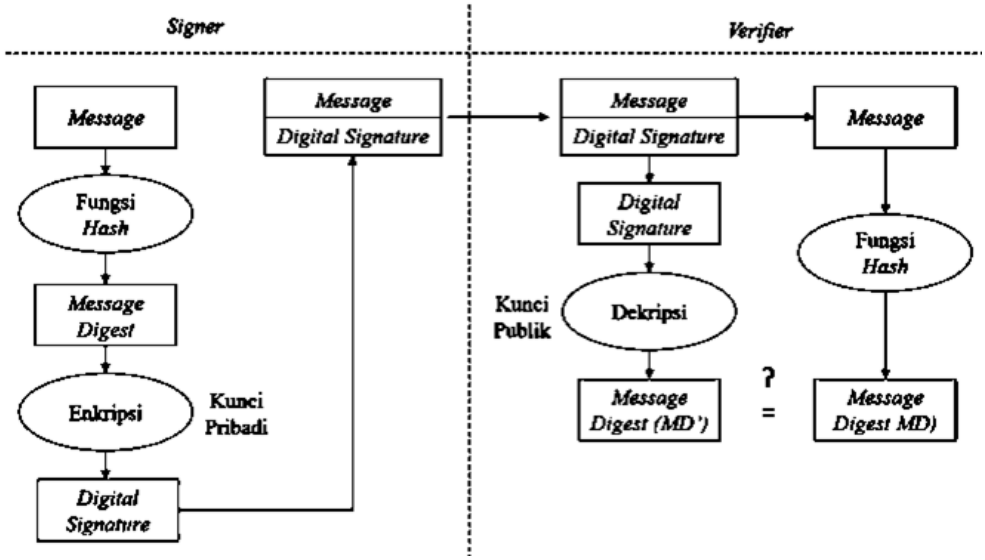
   Kriptografi kunci publik dan kriptografi kunci simetris dapat diterapkan dalam proses enkripsi pesan untuk memastikan keamanan. Dalam komunikasi terenkripsi, pesan dianggap secara implisit telah ditandatangani. Pada enkripsi simetris, diperlukan perantara untuk membagikan kunci rahasia secara aman antara pihak pengirim dan penerima. Sebaliknya, dalam kriptografi kunci asimetris, tidak diperlukan perantara, karena kunci publik dapat dibagikan secara terbuka, sementara kunci privat disimpan secara rahasia oleh pemiliknya.

2. Tanda Tangan Digital dengan Fungsi *Hash* dan Algoritma Kriptografi Kunci Publik

   Pesan dapat dienkripsi dengan kriptografi kunci publik, di mana algoritma kunci publik yang dipilih harus memenuhi persyaratan bahwa $D_d(E_e(M)) = M$ dan $D_e(E_d(M)) = M$, dengan e sebagai kunci publik dan d sebagai kunci privat. Setelah dienkripsi, pesan ditandai dengan string "*begin PGP Signature"* dan "*end PGP Signature"* menunjukkan bahwa

nilai *hash*, yang merangkum pesan, berfungsi sebagai tanda tangan digital.

Penggunaan kriptografi kunci publik untuk mengenkripsi *message digest* menghasilkan tanda tangan digital yang menjamin integritas pesan. Pesan asli yang tidak dienkripsi dikirim bersama dengan tanda tangan digital, yang kemudian dapat diverifikasi oleh penerima dengan membandingkan *message digest* yang dihasilkan ulang. Proses ini memastikan keaslian pesan dengan memadukan kriptografi kunci publik dan fungsi *hash*. Berikut gambar 2.2 yang menunjukkan skema proses tanda tangan digital yang mencakup langkah-langkah penting dalam verifikasi integritas dan autentikasi pesan



Gambar 2.2 Skema *Digital Signature*
(Sumber: https://images.app.goo.gl/K34yEG4WcqaMyRJW7)

Gambar 2.2 menunjukkan prosedur operasional tanda tangan digital yang terdiri dari beberapa langkah penting, yaitu:

a) Pengirim menghasilkan *message digest* dari dokumen yang akan dikirim dengan menggunakan teknik *hashing*.

b) Setelah *message digest* terbentuk, pengirim menandatangani *message*

digest tersebut menggunakan kunci privat untuk menghasilkan tanda tangan digital.

c) Pengirim kemudian mengirimkan dokumen kepada penerima beserta tanda tangan digital yang dilampirkan..

d) Penerima menerima pesan yang dikirim oleh pengirim.

e) Penerima melakukan proses verifikasi dengan menghitung *hash* dari pesan yang diterima untuk menghasilkan *message digest* baru. Selanjutnya, kunci publik pengirim digunakan untuk memverifikasi tanda tangan digital. Jika *message digest* yang dihasilkan sama dengan *message digest* yang diperoleh dari tanda tangan digital, komunikasi tersebut dianggap autentik dan berasal dari pengirim yang sah. Sebaliknya, jika terdapat perbedaan pada *message digest*, hal ini menandakan bahwa pesan telah diubah oleh pihak yang tidak bertanggung jawab.

2.5 SHA (*Secure Hash Algorithm)*

Algoritma fungsi *hash* satu arah yang dikembangkan oleh *National Security Agency* (NSA) dan dirilis oleh *National Institute of Standards and Technology* (NIST) dikenal sebagai *Secure Hash Algorithm* (SHA). Pada tahun 2005, ditemukan kerentanan dalam algoritma ini yang memungkinkan pembuatan *message digest* identik dengan kunci publik yang berbeda, sehingga membuat pendahulunya, MD5, tidak lagi dianggap aman. SHA merupakan pengembangan dari pendekatan tersebut, dirancang sedemikian rupa sehingga menemukan pesan yang sesuai dengan *message digest* tertentu tidak mungkin dilakukan secara komputasional. Oleh karena itu, algoritma ini tetap dianggap aman. Terdapat enam

varian SHA yang telah dirilis, yaitu SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, dan SHA-512 (Insani, 2008).

Sebagai varian yang paling banyak digunakan dan sering dibandingkan dengan SHA-2, SHA-256 akan menjadi fokus pembahasan selanjutnya. SHA-256 adalah bagian dari keluarga SHA-2, yang merupakan proyek yang dikembangkan oleh pemerintah Amerika Serikat dan menghasilkan empat variasi dalam kategori ini. Variasi tersebut meliputi SHA-224, SHA-384, SHA-512/224, dan SHA-512/256. SHA-512/256, yang kadang-kadang disebut sebagai SHA-256, adalah varian SHA-2 dengan panjang *hash* 256-bit. Dengan demikian, SHA-256 sering digunakan sebagai pengganti SHA-2 yang lebih umum, sehingga dapat dikatakan bahwa SHA-2 dan SHA-256 dapat dipertukarkan tanpa perbedaan mendasar (Rodriguez-Henriquez, 2006).

Metode *hashing* yang digunakan dalam SHA-256 memiliki hubungan erat dengan SHA-2. Ketika SHA-2 diterapkan dalam sistem yang sebanding, SHA-256 sering dijadikan alternatif. Pada dasarnya, SHA-256 adalah salah satu dari banyak fungsi *hash* yang tersedia, dan penerapannya yang efektif memerlukan pemahaman mendalam tentang algoritma tersebut. Meskipun demikian, baik SHA-256 maupun jenis fungsi *hash* lainnya tetap memiliki peranan penting dalam berbagai aplikasi keamanan data.

Beberapa fitur menonjol dari algoritma SHA adalah sebagai berikut:

1. Panjang Pesan: Panjang teks yang diinputkan harus kurang dari 264-bit. Ukurannya harus berada dalam batasan yang ditetapkan untuk menjaga agar intisari *hash* tetap acak.
2. Panjang Intisari: Panjang intisari *hash* harus 256-bit untuk SHA-256, 512-

bit untuk SHA-512, dan seterusnya. Intisari yang lebih besar biasanya memerlukan perhitungan yang lebih banyak, mengorbankan kecepatan dan ruang.

3. Ketidakberubahan: Secara desain, semua fungsi *hash*, termasuk SHA-256, tidak dapat diubah. Teks asli tidak dapat diperoleh dari intisari yang dihasilkan, dan intisari juga tidak dapat memberikan nilai aslinya saat melewati fungsi *hash.*

Berikut adalah tahapan dalam proses pembuatan *message digest* menggunakan SHA-256 (Mankar dan Nipanikar, 2013):

1. Pembuatan *padding bits* (bit-bit pengganjal) dari pesan.

   Langkah pertama dalam fungsi *hashing* adalah menambahkan bit-bit pengganjal ke pesan asli agar panjangnya sesuai dengan standar yang diperlukan untuk fungsi *hash*. Bit-bit tersebut ditambahkan sedemikian rupa sehingga panjang pesan menjadi tepat 64-bit kurang dari kelipatan 512. Penambahan ini dimulai dengan satu bit ‘1’, diikuti oleh sejumlah bit ‘0’ hingga panjangnya memenuhi kriteria tersebut. Persamaan yang digunakan untuk menghitung jumlah bit '0' pada padding adalah sebagai berikut:

$$l + 1 + k \equiv 448 \ (mod\ 512)$$

$$k \equiv 448 - (l + 1)(\ mod\ 512)$$

   Keterangan:

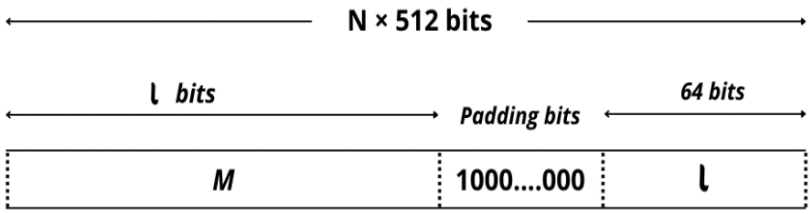
   a) l : panjang pesan asli pesan dalam bit.

   b) k : jumlah bit tambahan yang ditambahkan pada pesan dalam proses *padding*.

   Pemilihan panjang 448 bit dalam *padding* ini berfungsi untuk

memastikan bahwa pesan yang telah di *padding* berada dalam struktur blok 512 bit. Dengan demikian, 64 bit terakhir dalam blok ini akan berisi representasi panjag pesan asli dalam bit. Struktur *padding* ini dirancang untuk mempertahankan konsistensi dan keamanan hasil *hashing*, memastikan bahwa setiap perubahan sekecil apa pun pada pesan asli akan menghasilkan nilai *hash* yang sangat berbeda, sesuai dengan prinsip keamanan kriptografi yang kuat.

Berikut gambar 2.3 yang menunjukkan proses pembuatan *padding bits* dalam algoritma SHA-256. Penambahan *padding* ini esensial untuk memastikan panjang pesan sesuai dengan kelipatan 512 bit, sehingga fungsi *hash* dapat beroperasi dengan efisien.



Gambar 2.3 Pembuatan *Padding Bits*

Dengan penambahan *padding bits*, algoritma SHA-256 dapat menjaga konsistensi struktur pesan yang diolah. Langkah ini sangat penting dalam menghasilkan *message digest* yang aman dan akurat, seperti yang digambarkan dalam gambar tersebut.

2. Penambahan *length bit* (l)

Setelah penambahan bit padding ke pesan asli, langkah selanjutnya adalah menambahkan bit panjang sebesar 64-bit pada akhir pesan. Penambahan ini bertujuan untuk memastikan panjang keseluruhan pesan

menjadi kelipatan tepat 512 bit. Nilai 64-bit tersebut dihitung berdasarkan sisa panjang pesan asli (tanpa padding) terhadap modulus 2^{32}. Panjang pesan ini kemudian ditambahkan ke bit-bit padding untuk menghasilkan blok pesan dengan panjang yang sesuai.

3. Menginisialisasi *buffer* (Penetapan nilai awal)

Setelah blok pesan siap untuk diproses dalam perhitungan *hash* akhir, perlu dilakukan inisialisasi terhadap beberapa nilai awal. Nilai-nilai ini digunakan sebagai parameter dasar untuk mendukung tahap-tahap berikutnya dalam algoritma *hashing*. Berikut adalah nilai inisialisasi untuk delapan variabel yang digunakan dalam algoritma SHA-256:

$H_0^{(0)} = 6a09e667$

$H_1^{(0)} = bb67ae85$

$H_2^{(0)} = 3c6ef372$

$H_3^{(0)} = a54ff53a$

$H_4^{(0)} = 510e527f$

$H_5^{(0)} = 9b05688c$

$H_6^{(0)} = 1f83d9ab$

$H_7^{(0)} = 5be0cd19$

Dalam algoritma SHA-256, sejumlah 64 kunci perlu diinisialisasi dan disimpan dalam array mulai dari indeks $K[0]$ hingga $K[63]$. Kunci-kunci ini berperan penting dalam proses komputasi dan dihasilkan dari akar pangkat tiga dari bilangan prima pertama, yang kemudian dikonversi ke dalam bentuk heksadesimal. Kunci-kunci tersebut diatur dalam urutan yang

spesifik untuk memastikan integritas algoritma hashing dalam setiap langkah komputasi. Berikut adalah 64 kunci tersebut:

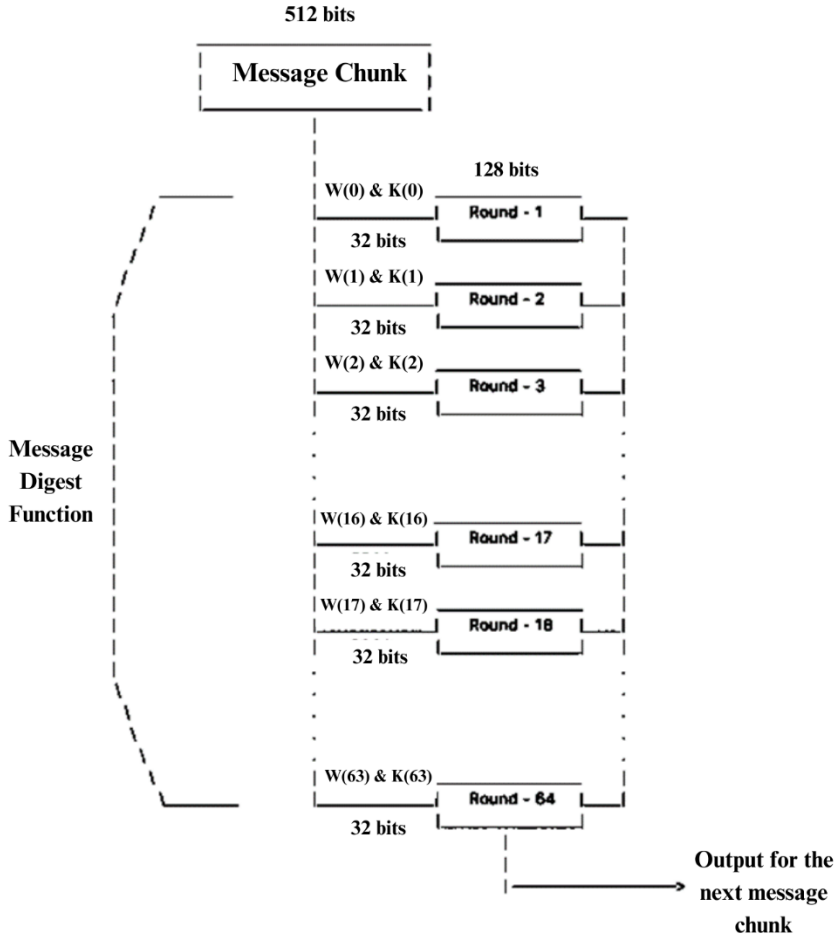
$k[0 \ldots 63] :=$

$0x428a2f98$, $0x71374491$, $0xb5c0fbcf$, $0xe9b5dba5$,
$0x3956c25b$, $0x59f111f1$, $0x923f82a4$, $0xab1c5ed5$,
$0xd807aa98$, $0x12835b01$, $0x243185be$, $0x550c7dc3$,
$0x72be5d74$, $0x80deb1fe$, $0x9bdc06a7$, $0xc19bf174$,
$0xe49b69c1$, $0xefbe4786$, $0x0fc19dc6$, $0x240ca1cc$,
$0x2de92c6f$, $0x4a7484aa$, $0x5cb0a9dc$, $0x76f988da$,
$0x983e5152$, $0xa831c66d$, $0xb00327c8$, $0xbf597fc7$,
$0xc6e00bf3$, $0xd5a79147$, $0x06ca6351$, $0x14292967$,
$0x27b70a85$, $0x2e1b2138$, $0x4d2c6dfc$, $0x53380d13$,
$0x650a7354$, $0x766a0abb$, $0x81c2c92e$, $0x92722c85$,
$0xa2bfe8a1$, $0xa81a664b$, $0xc24b8b70$, $0xc76c51a3$,
$0xd192e819$, $0xd6990624$, $0xf40e3585$, $0x106aa070$,
$0x19a4c116$, $0x1e376c08$, $0x2748774c$, $0x34b0bcb5$,
$0x391c0cb3$, $0x4ed8aa4a$, $0x5b9cca4f$, $0x682e6ff3$,
$0x748f82ee$, $0x78a5636f$, $0x84c87814$, $0x8cc70208$,
$0x90befffa$, $0xa4506ceb$, $0xbef9a3f7$, $0xc67178f2$

4. Fungsi Kompresi

Tahapan krusial dalam algoritma *hashing* terjadi pada langkah ini. Setiap blok pesan yang memiliki panjang $'n\ x\ 512'$ bit dibagi menjadi $'n'$ bagian, masing-masing berukuran 512 bit. Setiap bagian kemudian menjalani 64 putaran operasi, di mana keluaran dari setiap putaran

digunakan sebagai masukan untuk putaran berikutnya. Proses berulang ini memastikan bahwa setiap bit dalam blok pesan berkontribusi terhadap hasil akhir dari fungsi $hash$, meningkatkan kompleksitas dan keamanan dari algoritma hashing yang diterapkan. Seluruh prosesnya adalah sebagai berikut:



Gambar 2.4 Alur Pembentukan Word untuk *Prepare Message Schedule*
(Sumber: https://images.app.goo.gl/bq7AbqVZACsxyh2W9)

Pada gambar tersebut, terlihat dengan jelas bahwa 64 putaran operasi diterapkan pada pesan berukuran 512 bit. Dua input yang digunakan dalam setiap putaran $W(t)$ & $K(t)$. Pada 16 putaran pertama, pesan 512 bit dibagi menjadi 16 bagian, masing-masing berukuran 32 bit. Setelah itu, $W(t)$

dihitung untuk setiap langkah selanjutnya berdasarkan aturan tertentu, yang memungkinkan pembentukan blok-blok data tambahan yang diperlukan untuk putaran operasi selanjutnya. Proses ini meningkatkan kompleksitas dan keamanan algoritma hashing. Sebelum memasuki perhitungan nilai W_t, perlu dicatat bahwa proses ini mengikuti dua tahap yang berbeda tergantung pada rentang indeks t. Rumus untuk perhitungan W_t adalah sebagai berikut:

$$W_t = \begin{cases} M_t^{(i)}, & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16}\,, & 16 \le t \le 63 \end{cases}$$

Di mana,

a) $\sigma_0^{\{256\}}(W_{t-15}) = ROTR^7(W_{t-15}) \oplus ROTR^{18}(W_{t-15}) \oplus SHR^3(W_{t-15})$

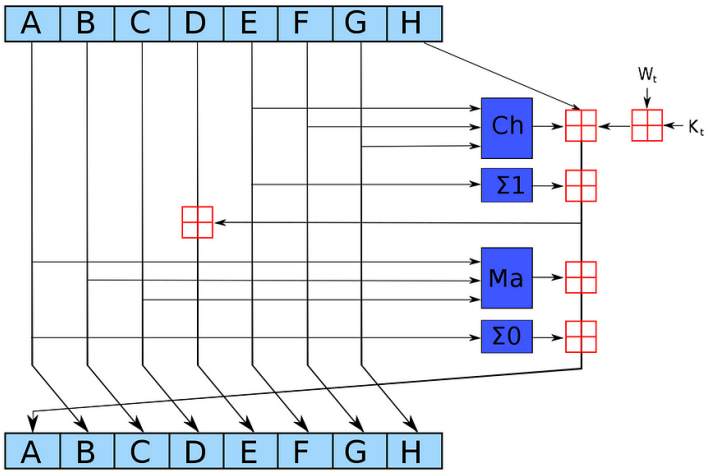
b) $\sigma_1^{\{256\}}(W_{t-2}) = ROTR^{17}(W_{t-2}) \oplus ROTR^{19}(W_{t-2}) \oplus SHR^{10}(W_{t-2})$

c) $ROTR^n(x)$ = Rotasi melingkar ke kanan dari $'x'$ sebanyak $'n'$ bit

d) $SHR^n(x)$ = Pergeseran melingkar ke kanan dari $'x'$ sebanyak $'n'$ bit

e) $\oplus$ = XOR (Eksklusif OR)

Berikut metode yang sudah teruji dalam membuat proses fungsi kompresi untuk masing-masing dari 64 putaran:



Gambar 2.5 Proses Fungsi Kompresi
(Sumber: https://images.app.goo.gl/1F4Rd3iG4hCZTkmm6)

Gambar di atas memperlihatkan secara jelas proses yang terjadi pada setiap putaran dalam algoritma SHA-256. Setelah nilai dan rumus untuk setiap fungsi ditetapkan, seluruh proses *hashing* dapat dijalankan. Fungsi logika utama yang diterapkan dalam masing-masing dari 64 putaran yang diulang adalah sebagai berikut:

a) $Ch(e, f, g) = (e \wedge f) \oplus ((\neg e) \wedge g)$

b) $Ma(a, b, c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$

c) $\sum_0^{\{256\}}(a) = ROTR^2(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a)$

d) $\sum_1^{\{256\}}(e) = ROTR^6(e) \oplus ROTR^{11}(e) \oplus ROTR^{25}(e)$

Keterangan: Simbol (¬) melambangkan operasi NOT, yang berfungsi untuk membalik nilai boolean. Jika *e* adalah 1 (*true*), maka ¬*e* akan menjadi 0 (*false*), dan sebaliknya. Sedangkan simbol (∧) melambangkan operasi AND, yang menghasilkan nilai *true* hanya jika kedua operandnya bernilai *true*. Dalam konteks ini, $e \wedge f$ akan bernilai *true* jika baik e maupun f adalah 1 (*true*).

Perlu dicatat bahwa rotasi penentu bisa menggunakan konstanta yang berbeda untuk SHA-512. Komponen yang diberi warna merah menunjukkan penjumlahann (adisi) dalam modulus 2^{32} untuk SHA-256, atau 2^{64} untuk SHA-512.

2.6 *Elliptic Curve Cryptography* (ECC)

Keamanan dalam kriptografi kurva eliptik (*Elliptic Curve Cryptography*/ECC) terletak pada kompleksitas matematika masalah logaritma diskret pada kurva eliptik atau *Elliptic Curve Discrete Logarithm Problem*

(ECDLP). Berbeda dengan masalah logaritma diskret (DLP) dan faktorisasi bilangan bulat (IFP), hingga saat ini ECDLP belum dapat diselesaikan dengan waktu subeksponensial, memberikan keunggulan berupa ukuran kunci yang lebih kecil namun dengan tingkat keamanan yang setara. Protokol yang menggunakan ECDLP meliputi *Elliptic Curve Diffie-Hellman* (ECDH), *Elliptic Curve ElGamal* (ECELGamal), dan *Elliptic Curve Digital Signature Algorithm* (ECDSA).

Elliptic Curve Cryptography sendiri merupakan teknik yang memanfaatkan struktur aljabar kurva eliptik pada bidang terbatas. Konsep ini pertama kali diajukan oleh Neal Koblitz dan Victor S. Miller pada 1985 dan kini juga diterapkan dalam teknik faktorisasi bilangan seperti metode Lenstra. Algoritma kunci publik seperti RSA mengandalkan varian komputasi kompleks dari dua bilangan prima besar untuk keamanan, di mana kunci publik dibagikan, sedangkan kunci privat dirahasiakan untuk memastikan otoritas eksklusif dalam dekripsi pesan.

Untuk membangun kriptografi kurva eliptik (*Elliptic Curve Cryptography*/ECC), dua titik pada kurva eliptik $E(F_p)$ dijumlahkan untuk menghasilkan titik ketiga pada kurva tersebut. Penjumlahan titik ini mengikuti aturan geometris yang dirancang khusus, di mana titik-titik pada kurva didefinisikan dalam bidang (F_p). Metode ini menjadi dasar operasi aritmatika pada kurva eliptik dan mendukung pembentukan kunci publik dan privat dalam sistem kriptografi ECC. Berikut ini adalah penjelasan geometris mengenai aturan tersebut:

$$y^2 = x^3 + ax + b + c \text{ dengan } 4a^3 + 27b^2 \neq 0$$

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \\ \dfrac{3x_1^2 + a}{2y_1} \end{cases}$$

Diberikan E oleh $y^2 = x^3 + ax + b + c$, $P_1 = (x_1, y_1)$ dan $P_2 = (x_2, y_2)$, $P_1 + P_2 = P_3(x_3, y_3)$ sebagai berikut:

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 + x_3) - y_1$$

Pada kriptografi kurva eliptik, titik P pada kurva eliptik dikalikan dengan bilangan skalar k melalui operasi perkalian titik, menghasilkan titik Q pada kurva yang sama sesuai persamaan $kP = Q$. Proses perkalian titik ini mencakup dua operasi utama, yaitu:

1. Operasi pertambahan titik. Artinya untuk memperoleh titik L, tambahkan dua titik yaitu J dan K, sehingga menghasilkan $L = J + K$.
2. Operasi penggandaan titik. Artinya untuk memperoleh titik L yaitu dengan menambahkan J dengan dirinya sendiri. Dengan demikian, $L = 2J$.

Setelah membahas dasar teori dan konsep *Elliptic Curve Cryptography* (ECC), selanjutnya akan dijelaskan implementasi praktisnya melalui *Elliptic Curve Digital Signature Algorithm* (ECDSA), yang merupakan salah satu teknik utama dalam menghasilkan tanda tangan digital berbasis kurva eliptik. ECDSA merupakan bagian dari *Digital Signature Standard* (DSS), yang terdiri dari dua komponen utama yaitu *Secure Hash Algorithm* (SHA) dan *Digital Signature Algorithm* (DSA). Dalam ECDSA, SHA berfungsi sebagai fungsi *hash*, sementara DSA menggunakan algoritma *Elliptic Curve Cryptography* (ECC). Keamanan ECDSA didasarkan pada kompleksitas penyelesaian masalah matematika kurva eliptik, dengan tiga langkah utama: tahap menentukan kunci, pembuatan tanda tangan, dan verifikasi tanda tangan (Triwinarko, 2005).

1. Tahap Menentukan *key*
   a) Memilih sebuah bilangan bulat random d_A, yang nilainya diantara $[1, n-1]$
   b) Menghitung $Q_A = d_A \cdot G = (x_1, y_1)$
   c) Kunci rahasia = d_A, dan kunci publik Q_A
2. Tahap Penandatanganan
   a) Memilih sebuah bilangan bulat random k yang nilainya diantara $[1, n-1]$
   b) Menghitung $R = k \cdot G = (x_1, y_1)$ dan $r = x_1 \ mod \ n$, jika $r = 0$, maka kembali ke langkah 1
   c) Menghitung $k^{-1} mod \ n$
   d) Menghitung $e = Hash(m)$
   e) Menghitung $s = k^{-1}\{e + d_A \cdot r\} \ mod \ n$

   Tanda tangan Alice untuk *message m* adalah (r, s)
3. Tahap Verifikasi
   a) Memverifikasi bahwa r dan s adalah bilangan bulat yang antara $[1, n-1]$
   b) Menghitung $e = Hash \ (m)$
   c) Menghitung $w = s^{-1} \ mod \ n$
   d) Menghitung $u_1 = e \cdot w \ mod \ n$ dan $u_2 = r \cdot w \ mod \ n$
   e) Menghitung $u_1 \cdot G + \ u_2 \cdot Q_A = (x_1, y_1)$
   f) Menghitung $v = x_1 \ mod \ n$ dan $z = y_1 \ mod \ n$
   g) Menerima tanda tangan jika dan hanya jika $v = r$

2.7 Tanda Tangan Digital Menggunakan SHA-256 dan ECDSA

Proses penandatanganan pesan dan verifikasi menggunakan fungsi *hash* SHA-256 dan algoritma ECDSA dapat dilakukan sebagai berikut:

1. Mengonversi pesan menjadi *message digest*:
    a) Konversi karakter ke biner: Setiap karakter dalam pesan diubah ke bentuk biner.
    b) Penambahan *padding bits*: Pada tahap ini, panjang pesan disesuaikan agar menjadi kelipatan 512 bit, dengan 64 bit terakhir dialokasikan untuk menyimpan panjang pesan asli. Prosesnya dimulai dengan menyisipkan bit ‘1’ setelah bit terakhir dari pesan asli, kemudian diikuti oleh sejumlah bit ‘0’ hingga panjang pesan memenuhi syarat ini. Proses ini memastikan pesan berada dalam format panjang yang diperlukan untuk pemrosesan lebih lanjut pada tahap *hashing*.
    c) Penambahan *length* bit: Menambahkan panjang pesan asli dalam bentuk 64-bit.
    d) Melakukan parsing pesan (pemecahan pesan): Hasil pesan yang di *padding* [M^0] dibagi menjadi 16 word berukuran 32-bit dan di konversi ke heksadesimal
    e) Inisialisasi *buffer* (penyangga): Pada SHA-256, menggunakan delapan variabel awal 32-bit yang diinisialisasi sebagai konstanta berikut:

    $H_0^{(0)}$ = $6a09e667$

    $H_1^{(0)}$ = $bb67ae85$

    $H_2^{(0)}$ = $3c6ef372$

    $H_3^{(0)}$ = $a54ff53a$

$H_4^{(0)}$ = $510e527f$

$H_5^{(0)}$ = $9b05688c$

$H_6^{(0)}$ = $1f83d9ab$

$H_7^{(0)}$ = $5be0cd19$

f) Pembentukan word untuk proses *hashing* atau *prepare message schedule*: Proses ini dilakukan untuk mendapatkan total 64 word W_t dengan menggunakan aturan berikut:

$$W_t = \begin{cases} M_t^{(i)}, & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16}, & 16 \le t \le 63 \end{cases}$$

Dengan fungsi logika utama yang telah di jelaskan di pembahasan sebelumya, diperoleh nilai $W_t^{(i)}$ sebagai hasil akhirnya.

g) Inisialisasi variabel kerja: Pada setiap putaran, variabel kerja $a, b, c, \dots, h$ diinisialisasi dari *buffer:*

$a = H_0^{(0)} = 6a09e667$

$b = H_1^{(0)} = bb67ae85$

$c = H_2^{(0)} = 3c6ef372$

$d = H_3^{(0)} = a54ff53a$

$e = H_4^{(0)} = 510e527f$

$f = H_5^{(0)} = 9b05688c$

$g = H_6^{(0)} = 1f83d9ab$

$h = H_7^{(0)} = 5be0cd19$

h) Proses kompresi 64 putaran:

- Menghitung T_1 dan T_2:

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$

$$T_2 = \sum_0^{\{256\}}(a) + Ma(a,b,c)$$

- Memperbarui variabel $a, b, c, \dots, h$:

$h = g$

$g = f$

$f = e$

$e = d + T_1$

$d = c$

$c = b$

$b = a$

$a = T_1 + T_2$

Langkah ini diulang hingga putaran ke-64 ($t = 63$).

i) Penjumlahan *buffer* akhir: Setelah semua putaran selesai, *buffer* SHA-256 diperbarui dengan menjumlahkan nilai *buffer* sesudah pembaruan (a') dengan nilai awal *buffer* (a), seperti berikut:

$H_0^{(0)} = a' + a$

$H_1^{(0)} = b' + \mathrm{b}$

$H_2^{(0)} = c' + c$

$H_3^{(0)} = d' + d$

$H_4^{(0)} = e' + e$

$H_5^{(0)} = f' + f$

$H_6^{(0)} = g' + g$

$H_7^{(0)} = h' + h$

Selanjutnya menggabungkan $H_0^{(0)} - H_7^{(0)}$ hingga diperoleh *message digest* (m).

j) *Message digest* (m) yang diperoleh tersusun dari hasil penggabungan nilai akhir setiap *buffer* dalam format heksadesimal. Untuk memahami nilai numeriknya dalam sistem desimal, dilakukan proses konversi bilangan heksadesimal ke desimal dengan langkah-langkah sebagai berikut:

- Mengidentifikasi setiap digit bilangan heksadesimal. Setiap digit dalam bilangan heksadesimal dicocokkan dengan nilai desimalnya, berdasarkan tabel konversi pada Tabel 2.1
- Menentukan posisi setiap digit dalam bilangan heksadesimal. Posisi digit ditentukan dari kanan ke kiri, dimulai dari indeks 0. Penomoran ini digunakan dalam perhitungan berikutnya.
- Mengalikan nilai desimal dari setiap digit dengan basis 16 yang dipangkatkan sesuai posisinya. Perhitungan dilakukan dengan menggunakan rumus berikut:

$$N = \sum_{i=0}^{n} d_i \times 16^i$$

Dengan:

- N adalah hasil konversi bilangan heksadesimal ke dalam bentuk desimal.

- d_i adalah nilai desimal dari digit heksadesimal pada posisi ke-i.
- i adalah posisi digit dari kanan ke kiri, dimulai dari 0.
- 16^i adalah basis heksadesimal yang dipangkatkan sesuai posisi digitnya.

- Menjumlahkan seluruh hasil perkalian. Setelah setiap digit dikalikan dengan 16 dipangkatkan posisi digitnya, hasilnya dijumlahkan untuk memperoleh nilai akhir dalam sistem desimal. Perhitungan ini dapat diperjelas dengan format berikut:

$$N = d_{63} \times 16^{63} + d_{62} \times 16^{62} + d_{61} \times 16^{61} + \cdots + d_0 \times 16^0$$

dengan d_i adalah nilai desimal dari masing-masing digit heksadesimal yang dikonversi.

2. Mengenkripsi *message digest* menggunakan algoritma ECDSA

a) Tahap penentuan kunci

- Memilih bilangan bulat acak d_A, dalam rentang $[1, n-1]$.
- Menghitung Q_A dengan rumus $Q_A = d_A \cdot G = (x_1, y_1)$, menghasilkan kunci privat (d_A) dan kunci publik (Q_A).

b) Tahap Penandatanganan

- Memilih k sebagai sebuah bilangan bulat acak dalam rentang $[1, n-1]$ dengan n yang telah ditentukan
- Menghitung $R = k \cdot G = (x_1, y_1)$ dan $r = x_1 \; mod \; n$
- Menghitung $k^{-1} mod \; n$ menggunakan algoritma *Euclidean Extended* yaitu $k \cdot k^{-1} \equiv 1 \; mod \; n$, sehingga mendapatkan hasil $k^{-1} \equiv k \; mod \; n$.
- Menghitung $e = Hash(m)$. Dalam proses enkripsi, nilai dari

message digest diubah dalam bentuk desimal.

- Menghitung s dengan rumus $s = k^{-1}\{e + d_A \cdot r\}\, mod\, n$. Tanda tangan pesan ini menghasilkan titik (r, s) sebagai *cipherteks*.

3. Tahap Verifikasi

a) Memverifikasi bahwa r dan s adalah bilangan bulat yang antara $[1, n-1]$

b) Menghitung $e = Hash\ (m)$

c) Menghitung $w = s^{-1}\ mod\ n$

d) Menghitung u_1 dan u_2 dengan rumus:

$u_1 = e \cdot w\ mod\ n$ dan $u_2 = r \cdot w\ mod\ n$

e) Menghitung titik (x_1, y_1) dengan menggunakan rumus:

$u_1 \cdot G + u_2 \cdot Q_A = (x_1, y_1)$

f) Menghitung v dengan menggunakan rumus $v = x_1\ mod\ n$ dan

$z = y_1\ mod\ n$

g) Menerima tanda tangan jika dan hanya jika $v = r$.

2.8 Amanah dalam Islam dan Relevensinya terhadap Keamanan Data

Dalam Kamus Besar Bahasa Indonesia (2016), Kementerian Pendidikan dan Kebudayaan mendefinisikan "amanah" sebagai sesuatu yang dipercayakan dan memiliki keandalan, keamanan, serta ketenteraman. Secara etimologis, kata "amanah" berasal dari bentuk *mashdar* dari kata kerja *amina-ya'manu-amnan-wa amanatan*, yang berarti aman, tenteram, dan damai. Sementara itu, Kamus Al-Munawwir mendefinisikan amanah sebagai segala perintah Allah SWT kepada hamba-hamba-Nya. Secara istilah, amanah memiliki makna luas yang mencakup dua aspek utama: pertama, sebagai tanggung jawab yang harus dijalankan dengan

jujur, dan kedua, sebagai titipan yang harus dijaga serta dikembalikan kepada pemiliknya. Kedua pengertian ini menggambarkan amanah sebagai kewajiban moral dan agama yang mencakup pengelolaan titipan dari Allah SWT serta pelaksanaan tugas dengan kejujuran dan integritas (Fachruddin Hs., Ensiklopedia Pengetahuan Al-Qur'an dan Hadits, hlm. 15). Hal ini sejalan dengan firman Allah SWT dalam QS. An-Nisa' ayat 58 yang menekankan pentingnya menyampaikan amanah kepada pihak yang berhak.

ٱللَّهَ نعِمَّا إِنَّ ٱللَّهَ يَأۡمُرُكُمۡ أَن تُؤَدُّواْ ٱلۡأَمَٰنَٰتِ إِلَىٰٓ أَهۡلِهَا وَإِذَا حَكَمۡتُم بَيۡنَ ٱلنَّاسِ أَن تَحۡكُمُواْ بِٱلۡعَدۡلِۚ إِنَّ يَعِظُكُم بِهِۦٓۗ إِنَّ ٱللَّهَ كَانَ سَمِيعَۢا بَصِيرٗا (النساء: ٥٨)

Artinya: "Sesungguhnya Allah SWT menyuruh kamu menyampaikan amanah kepada yang berhak menerimanya, dan (menyuruh kamu) apabila menetapkan hukum di antara manusia supaya kamu menetapkan dengan adil. Sesungguhnya Allah SWT memberi pengajaran yang sebaik-baiknya kepadamu. Sesungguhnya Allah SWT adalah Maha Mendengar lagi Maha Melihat". (An-Nisa':58)

Ayat tersebut secara jelas menunjukkan perintah Allah SWT untuk menyampaikan amanah kepada pihak yang berhak menerimanya. Setiap individu yang diberikan amanah dalam menengahi perselisihan wajib mengambil keputusan dengan adil. Prinsip ini esensial karena Allah SWT Maha Mendengar dan Maha Melihat segala perbuatan hamba-Nya, terutama dalam upaya menjalankan perintah-perintah-Nya.

Ibnu Mardawaih meriwayatkan dari al-Kalbi melalui Abu Shaleh bahwa Ibnu Abbas menyampaikan, ketika Rasulullah SAW berhasil menaklukkan Mekah, beliau memanggil Utsman bin Thalhah. Saat Utsman datang, Rasulullah SAW meminta kunci Ka'bah darinya. Utsman kemudian pergi mengambil kunci tersebut dan menyerahkannya kepada Rasulullah SAW dengan telapak tangan terbuka. Pada saat itu, al-Abbas berdiri dan memohon kepada Rasulullah SAW agar diberikan

kunci tersebut sehingga ia bisa mengemban tugas menjaga kunci Ka'bah sekaligus menyediakan minuman. Namun, Utsman tetap memegang kunci tersebut. Rasulullah SAW meminta kembali kunci tersebut, dan Utsman menyerahkannya dengan berkata, “Terimalah dengan amanah Allah.” Rasulullah SAW kemudian membuka pintu Ka'bah dan melakukan thawaf. Tak lama kemudian, Jibril turun menyampaikan wahyu yang memerintahkan Rasulullah SAW untuk mengembalikan kunci itu kepada Utsman bin Thalhah. Rasulullah SAW memanggil Utsman dan mengembalikan kunci tersebut sambil membacakan firman Allah SWT sebagai berikut:

> “Sungguh, Allah memerintahkan kamu untuk menyampaikan amanat kepada yang berhak menerimanya...” (An-Nisa’: 58) (Shaleh, 2009).

Menurut penafsiran Al-Misbah, amanah merupakan sesuatu yang dipercayakan kepada seseorang atau pihak ketiga untuk dijaga dan dikembalikan sesuai dengan kehendak pemiliknya. Amanah ini bertentangan dengan sifat khianat dan diberikan kepada individu yang dipandang memiliki kemampuan serta keandalan oleh pemberi amanah, sehingga mereka diharapkan mampu menjaga dengan baik hal-hal yang telah dipercayakan tersebut.

Berdasarkan sabda Nabi SAW, "Tidak ada iman bagi mereka yang tidak memiliki amanah," agama mengajarkan bahwa amanah atau kepercayaan merupakan manifestasi dari keimanan seseorang. Lebih dari itu, setiap tindakan dan interaksi mencerminkan kualitas amanah yang dimiliki seseorang, yang menjadi kebalikan dari sifat khianat. Keberadaan amanah menjadi syarat agar seseorang dapat dipercaya, serta membawa ketenangan batin yang akan memperkuat keyakinan dalam dirinya. Pandangan ini menunjukkan bahwa amanah mencakup aspek non-material maupun material, yang menuntut pemenuhan setiap tanggung

jawab sesuai dengan ketentuan yang semestinya, sebagaimana diperintahkan oleh Allah SWT.

Amanah, dalam beberapa konteks, dapat dipandang sebagai sesuatu yang berasal dari Allah SWT. Namun, dari perspektif lain, amanah ini juga dipercayakan kepada berbagai makhluk, termasuk malaikat, jin, dan manusia, baik mereka yang berstatus nabi maupun bukan. Ketaatan kepada Allah SWT dan Rasul-Nya merupakan kewajiban yang sangat penting; mereka yang lalai dalam ketaatan akan menanggung konsekuensinya, sedangkan yang setia dalam kepatuhan akan meraih kemenangan besar. Ini menunjukkan betapa menantangnya bagi manusia untuk mematuhi syariat yang ketat, menggambarkan tanggung jawab besar dalam menjaga amanah yang diberikan.

Amanah mencakup berbagai aspek hubungan, termasuk hubungan antara manusia dengan Tuhan, antar sesama manusia, lingkungan, dan juga diri sendiri. Setiap amanah, bahkan yang melibatkan individu secara pribadi, mengandung tanggung jawab yang harus dipenuhi dengan sepenuh hati. Amanah tidak hanya terbatas pada kewajiban material, tetapi juga meliputi tanggung jawab moral dan spiritual, sehingga mencakup baik kewajiban non-material yang bersifat moral maupun material yang bersifat praktis.

Ayat 58 Surah An-Nisa mengandung perintah agar amanah diserahkan kepada pihak yang berhak menerima. Ayat ini juga menekankan kewajiban untuk berlaku adil dalam menetapkan hukum, tanpa membedakan ras, agama, atau keturunan. Penekanan ini memperlihatkan pentingnya menegakkan keadilan dan amanah secara setara. Prinsip keadilan ini juga ditegaskan di berbagai bagian lain dalam Al-Qur'an, salah satunya melalui peringatan kepada Nabi Muhammad SAW

ketika beliau hampir terpedaya oleh tipu daya seorang Muslim munafik yang berusaha menyalahkan seorang Yahudi.

Islam menekankan pentingnya keadilan dalam setiap keputusan yang diambil. Nabi Muhammad SAW diperingatkan agar tidak membela pengkhianat, meskipun mereka berstatus sebagai Muslim. Hal ini ditegaskan dalam Surah An-Nisa' ayat 105:

إِنَّا أَنْزَلْنَا إِلَيْكَ الْكِتٰبَ بِالْحَقِّ لِتَحْكُمَ بَيْنَ النَّاسِ بِمَا أَرٰىكَ اللّٰهُ وَلَا تَكُنْ لِّلْخَآئِنِيْنَ خَصِيْمًا (النساء: ١٠٥)

Artinya: “Sesungguhnya Kami telah menurunkan Kitab (Al-Qur’an) kepadamu (Nabi Muhammad) dengan hak agar kamu memutuskan (perkara) di antara manusia dengan apa yang telah Allah SWT ajarkan kepadamu. Janganlah engkau menjadi penentang (orang yang tidak bersalah) karena (membela) para pengkhianat” (QS. An-Nisa’:105).

Berdasarkan penjelasan tersebut, dapat disimpulkan bahwa menurut tafsir Al-Misbah, amanah adalah sesuatu yang dipercayakan kepada seseorang untuk dijaga dan kemudian dikembalikan kepada pemiliknya (Q. Shaleh dkk., 2009).

Rasulullah SAW bersabda, "Tanda-tanda orang munafik ada empat: (1) jika berbicara, ia berdusta; (2) jika berjanji, ia tidak menepati; (3) jika berdebat, ia berpaling dari kebenaran; (4) jika membuat perjanjian, ia mengkhianatinya" (HR. Bukhari dan Muslim). Setiap individu yang memiliki salah satu sifat tersebut tergolong munafik, namun tidak serta merta menyebabkan kekafiran, melainkan merugikan dirinya sendiri (Hamidy, 1981).

BAB III

METODE PENELITIAN

3.1 Jenis Penelitian

Penelitian ini menggunakan metode campuran *(mixed methods)*, yang menggabungkan pendekatan kualitatif dan kuantitatif. Pendekatan kualitatif digunakan untuk menganalisis teori kriptografi, fungsi *hash* SHA-256, dan algoritma ECDSA berdasarkan studi literatur. Sementara itu, pendekatan kuantitatif diterapkan untuk menguji keakuratan tanda tangan digital yang dihasilkan dengan menggunakan 30 dokumen PDF sebagai sampel pengujian.

3.2 Subjek dan Objek Penelitian

Subjek dalam penelitian ini adalah algoritma SHA-256 dan ECDSA yang digunakan dalam implementasi tanda tangan digital. Sedangkan objek penelitian berupa aplikasi tanda tangan digital yang dikembangkan serta hasil tanda tangan digital yang diuji menggunakan dokumen PDF.

3.3 Metode Pengumpulan Data

Metode pengumpulan data dilakukan melalui dua pendekatan, yaitu:

1. Pendekatan Kualitatif
    a) Mengkaji literatur dari jurnal, buku, dan artikel ilmiah yang membahas tanda tangan digital, fungsi *hash* SHA-256, dan algoritma ECDSA.
    b) Menganalisis konsep serta aturan standar kriptografi yang digunakan dalam implementasi tanda tangan digital.

2. Pendekatan Kuantitatif
    a) Melakukan pengujian aplikasi dengan 30 dokumen PDF untuk mengukur akurasi tanda tangan digital.
    b) Menganalisis hasil tanda tangan digital dan proses verifikasinya berdasarkan nilai *hash*, kunci privat, kunci publik, serta parameter ECDSA (r, s).

3.4 Metode Pengolahan dan Analisis Data

Data yang diperoleh dari penelitian ini diolah dan dianalisis dengan menggunakan dua pendekatan sebagai berikut:

1. Pendekatan Kualitatif
    a) Menganalisis validitas algoritma SHA-256 dan ECDSA berdasarkan teori dan studi literatur.
    b) Mengevaluasi kesesuaian perhitungan manual dengan hasil implementasi algoritma pada aplikasi.
2. Pendekatan Kuantitatif
    a) Menghitung nilai *hash* serta parameter tanda tangan digital (r, s) dari setiap dokumen PDF yang diuji.
    b) Membandingkan hasil tanda tangan digital dengan hasil perhitungan manual untuk mengevaluasi tingkat keakuratan sistem.

3.5 Tahap-tahap Penelitian

Tahapan-tahapan dalam penelitian ini dilakukan sebagai berikut:

1. Perhitungan Manual

   Perhitungan manual dilakukan untuk memahami proses pembentukan tanda tangan digital sebelum algoritma diimplementasikan dalam aplikasi. Proses ini diawali dengan mengonversi pesan menjadi *message digest* menggunakan fungsi *hash* SHA-256. Selanjutnya, dilakukan proses enkripsi *message digest*, yang mencakup penentuan titik-titik pada kurva eliptik, pembentukan kunci privat dan publik, serta pembuatan tanda tangan digital melalui perhitungan parameter r dan s sesuai dengan algoritma ECDSA. Terakhir tanda tangan digital yang telah dihasilkan diverifikasi menggunakan kunci publik untuk memastikan keabsahannya.

   Setiap langkah dalam proses ini akan dijelaskan secara sistematis untuk menunjukkan bagaimana algoritma SHA-256 dan ECDSA bekerja dalam menghasilkan tanda tangan digital yang valid, sebagai berikut:

   a) Mengonversi pesan menjadi *message digest* menggunakan SHA-256, dengan tahapan sebagai berikut:

      - Penambahan *padding bits*: Pada tahap ini, panjang pesan disesuaikan agar menjadi kelipatan 512 bit, dengan 64 bit terakhir dialokasikan untuk menyimpan panjang pesan asli. Prosesnya dimulai dengan menyisipkan bit ‘1’ setelah bit terakhir dari pesan asli, kemudian diikuti oleh sejumlah bit ‘0’ hingga panjang pesan memenuhi syarat ini. Proses ini memastikan pesan berada dalam format panjang yang diperlukan untuk pemrosesan lebih lanjut

pada tahap *hashing*.

- Penambahan *length* bit: Menambahkan panjang pesan asli dalam bentuk 64-bit.
- Melakukan parsing pesan (pemecahan pesan): Hasil pesan yang di *padding* [M^0] dibagi menjadi 16 word berukuran 32-bit dan di konversi ke heksadesimal
- Inisialisasi *buffer* (penyangga): Pada SHA-256, menggunakan delapan variabel awal 32-bit yang diinisialisasi sebagai konstanta berikut:

  $H_0^{(0)}$ = $6a09e667$

  $H_1^{(0)}$ = $bb67ae85$

  $H_2^{(0)}$ = $3c6ef372$

  $H_3^{(0)}$ = $a54ff53a$

  $H_4^{(0)}$ = $510e527f$

  $H_5^{(0)}$ = $9b05688c$

  $H_6^{(0)}$ = $1f83d9ab$

  $H_7^{(0)}$ = $5be0cd19$

- Pembentukan word untuk proses *hashing* atau *prepare message schedule*: Proses ini dilakukan untuk mendapatkan total 64 word W_t dengan menggunakan aturan berikut:

$$W_t = \begin{cases} M_t^{(i)}, & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16}, & 16 \le t \le 63 \end{cases}$$

Dengan fungsi logika utama yang telah di jelaskan di pembahasan

sebelumya, diperoleh nilai $W_t^{(i)}$ sebagai hasil akhirnya.

- Inisialisasi variabel kerja: Pada setiap putaran, variabel kerja $a, b, c, \dots, h$ diinisialisasi dari *buffer:*

  $a = H_0^{(0)} = 6a09e667$

  $b = H_1^{(0)} = bb67ae85$

  $c = H_2^{(0)} = 3c6ef372$

  $d = H_3^{(0)} = a54ff53a$

  $e = H_4^{(0)} = 510e527f$

  $f = H_5^{(0)} = 9b05688c$

  $g = H_6^{(0)} = 1f83d9ab$

  $h = H_7^{(0)} = 5be0cd19$

- Proses kompresi 64 putaran:

  - Menghitung T_1 dan T_2:

    $$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e, f, g) + K_t^{\{256\}} + W_t$$

    $$T_2 = \sum_0^{\{256\}}(a) + Maj(a, b, c)$$

  - Memperbarui variabel $a, b, c, \dots, h$:

    $h = g$

    $g = f$

    $f = e$

    $e = d + T_1$

    $d = c$

    $c = b$

$b = a$

$a = T_1 + T_2$

Langkah ini diulang hingga putaran ke-64 ($t = 63$).

- Penjumlahan *buffer* akhir: Setelah semua putaran selesai, *buffer* SHA-256 diperbarui dengan menjumlahkan nilai *buffer* sesudah pembaruan (a') dengan nilai awal *buffer* (a), seperti berikut:

  $H_0^{(0)} = a' + a$

  $H_1^{(0)} = b' + \mathrm{b}$

  $H_2^{(0)} = c' + c$

  $H_3^{(0)} = d' + d$

  $H_4^{(0)} = e' + e$

  $H_5^{(0)} = f' + f$

  $H_6^{(0)} = g' + g$

  $H_7^{(0)} = h' + h$

  Selanjutnya menggabungkan $H_0^{(0)} - H_7^{(0)}$ hingga diperoleh *message digest* (m).

- *Message digest* (m) yang diperoleh tersusun dari hasil penggabungan nilai akhir setiap *buffer* dalam format heksadesimal. Untuk memahami nilai numeriknya dalam sistem desimal, dilakukan proses konversi bilangan heksadesimal ke desimal dengan langkah-langkah sebagai berikut:

  - Mengidentifikasi setiap digit bilangan heksadesimal. Setiap

digit dalam bilangan heksadesimal dicocokkan dengan nilai desimalnya.

- Menentukan posisi setiap digit dalam bilangan heksadesimal. Posisi digit ditentukan dari kanan ke kiri, dimulai dari indeks 0. Penomoran ini digunakan dalam perhitungan berikutnya.
- Mengalikan nilai desimal dari setiap digit dengan basis 16 yang dipangkatkan sesuai posisinya. Perhitungan dilakukan dengan menggunakan rumus berikut:

$$N = \sum_{i=0}^{n} d_i \times 16^i$$

dengan d_i merupakan nilai desimal dari digit heksadesimal pada posisi ke-i, dan 16^i menunjukkan pangkat basis sesuai dengan posisi digitnya. Konversi nilai digit heksadesimal ke desimal dilakukan berdasarkan Tabel ASCII.

- Menjumlahkan seluruh hasil perkalian. Setelah setiap digit dikalikan dengan 16 dipangkatkan posisi digitnya (16^i), hasilnya dijumlahkan untuk memperoleh nilai akhir dalam sistem desimal. Perhitungan ini dapat diperjelas dengan format berikut:

$$N = d_{63} \times 16^{63} + d_{62} \times 16^{62} + d_{61} \times 16^{61} + \cdots + d_0 \times 16^0$$

dengan d_i adalah nilai desimal dari masing-masing digit heksadesimal yang dikonversi.

b) Mengenkripsi *message digest* menggunakan algoritma enkripsi kriptografi ECDSA, dengan tahapan sebagai berikut:

- Penentuan titik kurva eliptik

  Tahap ini melibatkan pemilihan titik-titik pada kurva eliptik yang memenuhi persamaan $y^2 \equiv x^3 + ax + b\ (mod\ p)$, di mana $a, b,$ dan p adalah parameter yang mendefinisikan kurva eliptik. Titik-titik yang dihasilkan akan menjadi basis dalam pembentukan kunci.

- Pembentukan kunci

  Pada tahap ini, kunci privat (d_A) dan kunci publik Q_A dihasilkan. Kunci privat (d_A) adalah bilangan acak dalam selang $[1, n-1]$, dengan n sebagai bilangan prima yang besar. Kunci publik dihitung menggunakan rumus $Q_A = d_A \cdot G = (x_1, y_1)$. Di mana G adalah titik generator pada kurva.

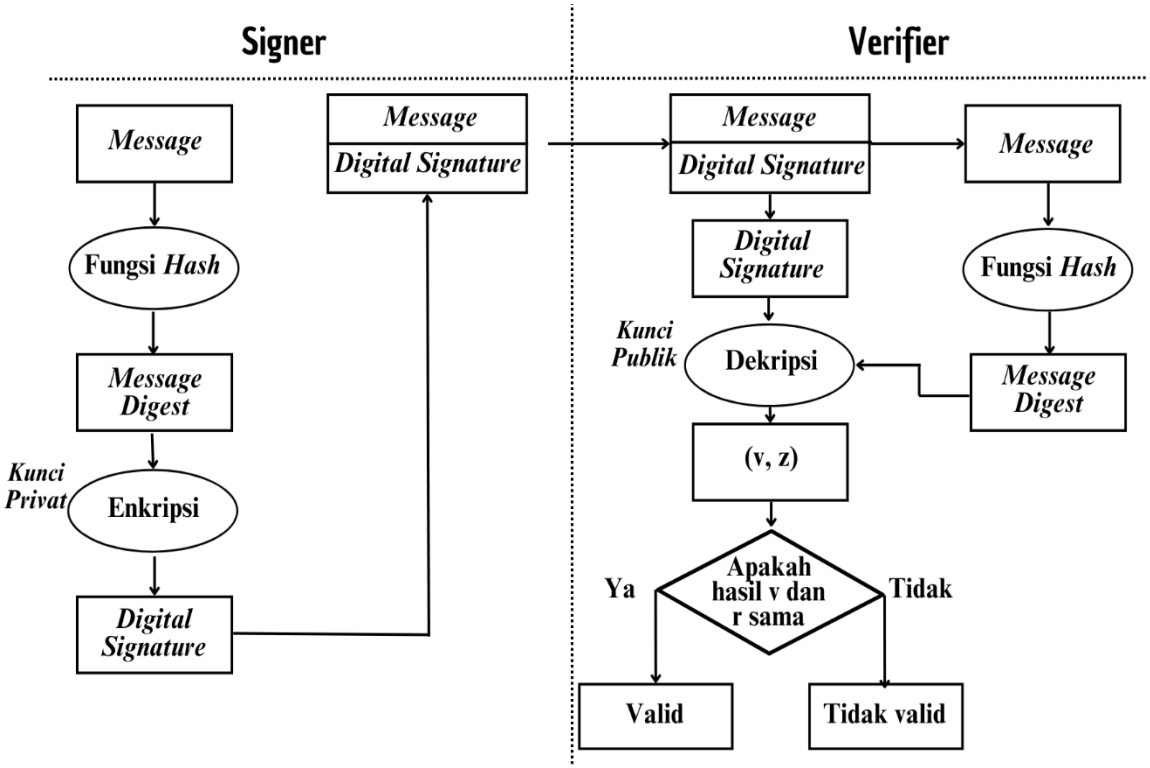
- Penandatanganan pesan
  - Memilih bilangan bulat acak k yang nilainya diantara $[1, n-1]$.
  - Menghitung titik $R = k \cdot G = (x_1, y_1)$ dan menentukan
  - $r = x_1\ mod\ n$, jika $r = 0$, maka pilih k baru.
  - Menghitung $k^{-1} mod\ n$ menggunakan algoritma *Euclidean Extended* yaitu $k \cdot k^{-1} \equiv 1\ mod\ n$, sehingga mendapatkan hasil $k^{-1} \equiv k\ mod\ n$
  - Menghitung $s = k^{-1}\{e + d_A \cdot r\}\ mod\ n$

c) Verifikasi tanda tangan (r, s) menggunakan fugsi *hash* SHA-256 dan algoritma dekripsi kriptografi ECDSA, dengan tahapan sebagai berikut:

- Memverifikasi bahwa r dan s berada dalam rentang $[1, n-1]$
- Menghitung $e = Hash\ (m)$

- Menghitung $w = s^{-1}\ mod\ n$, untuk mencari s^{-1} dapat dilakukan dengan menggunakan algoritma *Euclidean Extended* yaitu $s \cdot s^{-1} \equiv 1\ mod\ n$, sehingga mendapatkan hasil $s^{-1} \equiv s\ mod\ n$
- Menghitung $u_1 = e \cdot w\ mod\ n$ dan $u_2 = r \cdot w\ mod\ n$
- Menentukan $u_1 \cdot G + u_2 \cdot Q_A = (x_1, y_1)$ dan menghitung $v = x_1\ mod\ n$ dan $z = y_1\ mod\ n$
- Verifikasi berhasil jika $v = r$, memastikan bahwa dokumen asli dan valid.

Berikut ini adalah *flowchart* yang menjelaskan tahapan penandatanganan dan verifikasi secara sistematis:



Gambar 3.1 *Flowchart* Tahapan Penandatanganan dan Verifikasi

2. Pengujian dan Analisis Data

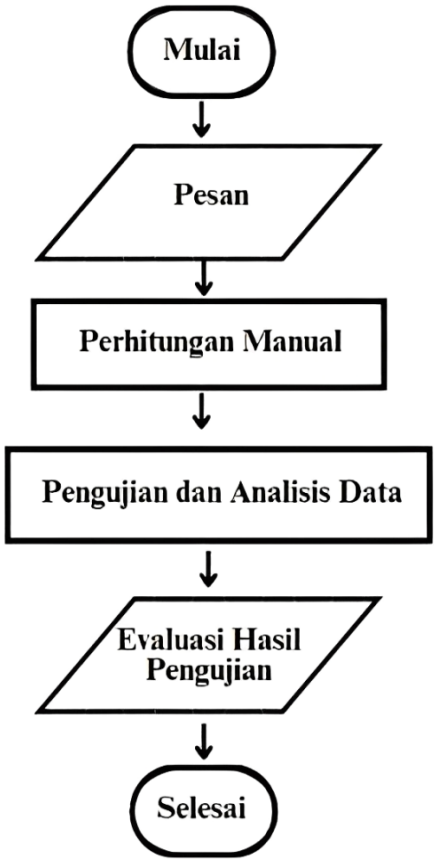
Bagian ini menjelaskan proses pengujian sistem tanda tangan digital berbasis SHA-256 dan ECDSA yang telah dikembangkan. Pengujian

bertujuan untuk memastikan bahwa sistem beroperasi sesuai dengan spesifikasi yang telah dirancang. Pengujian dilakukan terhadap 30 dokumen PDF untuk menguji keakuratan hasil tanda tangan digital, validitas proses verifikasi, serta performa sistem dalam menghasilkan dan memverifikasi tanda tangan digital. Proses pengujian dilakukan dengan menerapkan berbagai skenario masukan, seperti variasi parameter kurva eliptik, perbedaan ukuran dokumen, serta pengujian terhadap manipulasi data guna mengukur ketahanan tanda tangan digital terhadap perubahan yang tidak sah. Setelah pengujian selesai, data yang diperoleh dianalisis untuk mengevaluasi efektivitas algoritma yang digunakan serta mengidentifikasi potensi perbaikan dalam sistem.

3. Evaluasi Hasil Pengujian

Bagian ini menjelaskan langkah-langkah dalam penyusunan kesimpulan berdasarkan hasil pengujian sistem. Evaluasi dilakukan dengan membandingkan hasil tanda tangan digital yang dihasilkan dengan nilai yang seharusnya sesuai dengan perhitungan algoritma ECDSA. Selain itu, evaluasi juga mencakup analisis terhadap kecepatan proses hashing, penandatanganan, dan verifikasi, serta tingkat kemudahan penggunaan aplikasi bagi pengguna. Kesimpulan yang diperoleh dalam penelitian ini menentukan keberhasilan sistem dalam memenuhi tujuan penelitian serta mengidentifikasi aspek yang masih perlu diperbaiki untuk pengembangan lebih lanjut.

Berikut disajikan *flowchart* kerangka penelitian yang memaparkan tahapan utama penelitian ini secara runtut dari awal hingga akhir:



Gambar 3.2 *Flowchart* Kerangka Penelitian

BAB IV

PEMBAHASAN

4.1 Pembuatan dan Verifikasi Tanda Tangan Digital Secara Manual

Sebelum dilakukan pengujian menggunakan aplikasi, pada subbab ini akan dilakukan perhitungan manual untuk menghasilkan tanda tangan digital dari suatu pesan sederhana. Proses ini bertujuan untuk memahami secara mendalam langkah-langkah dasar dalam penerapan algoritma *Elliptic Curva Digital Signature* (ECDSA) dan fungsi *hash* SHA-256. Perhitungan dan verifikasi manual ini akan mencakup tahapan-tahapan berikut:

1. Mengonversi pesan menjadi *message digest*:
    a) Konversi karakter ke biner: Setiap karakter dalam pesan diubah ke bentuk biner berdasarkan Tabel ASCII, sebagaimana tercantum pada Lampiran 1. Kata yang digunakan adalah “KRIPTOGRAFI”
        - ‘K’ = 01001011
        - ‘R’ = 01010010
        - ‘I’ = 01001001
        - ‘P’ = 01010000
        - ‘T’ = 01010100
        - ‘O’ = 01001111
        - ‘G’ = 01000111
        - ‘R’ = 01010010
        - ‘A’ = 01000001
        - ‘F’ = 01000110

- ‘I’ = 01001001

Sehingga, “KRIPTOGRAFI” dalam bentuk biner yaitu:

01001011010100100100100101010000010101000100111101000111
01010010010000010100011001001001

b) Penambahan *padding bits*: Pada tahap ini, panjang pesan disesuaikan agar menjadi kelipatan 512 bit, dengan 64 bit terakhir dialokasikan untuk menyimpan panjang pesan asli. Prosesnya dimulai dengan menyisipkan bit ‘1’ setelah bit terakhir dari pesan asli, kemudian diikuti oleh sejumlah bit ‘0’ hingga panjang pesan memenuhi syarat ini.

$M = 01001011\ 01010010\ 01001001\ 01010000$
$01010100\ 01001111\ 01000111\ 01010010$
$01000001\ 01000110\ 01001001\ 10000000$

Panjang pesan yaitu: $l = 88$ bit

Untuk mencari nilai k, maka:

$k \equiv 448 - (l + 1)(\ mod\ 512)$

$k \equiv 448 - (88 + 1)(\ mod\ 512)$

$k = 359\ (mod\ 512)$

Maka, banyak bit $'0'$ yang ditambahkan sejumlah 359-bit

c) Penambahan *length* bit: Menambahkan panjang pesan asli dalam bentuk 64-bit. Panjang asli pesan adalah 88-bit (11 karakter × 8-bit) = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 01011000 (dalam bentuk biner). Maka dapat ditulis sebagai berikut:

01001011 01010010 01001001 01010000

01010100 01001111 01000111 01010010

01000001 01000110 01001001 10000000

⋮

00000000 00000000 00000000 01011000

d) Melakukan parsing pesan (pemecahan pesan): Hasil pesan yang di *padding* [M^0] dibagi menjadi 16 word berukuran 32-bit dan di konversi ke heksadesimal, seperti berikut:

$M_0^{(0)}$: 0100 1011 0101 0010 0100 1001 0101 0000

$M_1^{(0)}$: 0101 0100 0100 1111 0100 0111 0101 0010

$M_2^{(0)}$: 0100 0001 0100 0110 0100 1001 1000 0000

⋮

$M_{15}^{(0)}$: 0000 0000 0000 0000 0000 0000 0101 1000

Jika di konversi ke heksadesimal, maka setiap word berukuran 32-bit yang diperoleh dari hasil parsing pesan yang telah di pisahkan menjadi 16 word diubah dari representasi biner ke representasi heksadesimal, sehingga diperoleh:

$M_0^{(0)}$: $4B52A950$

$M_1^{(0)}$: $544F4752$

$M_2^{(0)}$: 41464980

⋮

$M_{15}^{(0)}$: 00000058

e) Inisialisasi *buffer* (penyangga): Pada SHA-256, menggunakan delapan variabel awal 32-bit yang diinisialisasi sebagai konstanta berikut:

$H_0^{(0)}$ = $6a09e667$

$H_1^{(0)}$ = $bb67ae85$

$H_2^{(0)}$ = $3c6ef372$

$H_3^{(0)}$ = $a54ff53a$

$H_4^{(0)}$ = $510e527f$

$H_5^{(0)}$ = $9b05688c$

$H_6^{(0)}$ = $1f83d9ab$

$H_7^{(0)}$ = $5be0cd19$

f) Pembentukan word untuk proses *hashing* atau *prepare message schedule*: Proses ini dilakukan untuk mendapatkan total 64 word W_t dengan menggunakan aturan berikut:

$$W_t = \begin{cases} M_t^{(i)}, & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16}, & 16 \le t \le 63 \end{cases}$$

Dengan fungsi logika utama yang telah di jelaskan di pembahasan sebelumya, diperoleh nilai $W_t^{(i)}$ sebagai berikut:

$W_0^{(0)}$: $4B52A950$

$W_1^{(0)}$: $544F4752$

$W_2^{(0)}$: 41464980

⋮

$W_{15}^{(0)}$: 00000058

g) Inisialisasi variabel kerja: Pada setiap putaran, variabel kerja $a, b, c, \dots, h$ diinisialisasi dari *buffer*, seperti berikut*:*

$a = H_0^{(0)} = 6a09e667$

$b = H_1^{(0)} = bb67ae85$

$c = H_2^{(0)} = 3c6ef372$

$d = H_3^{(0)} = a54ff53a$

$e = H_4^{(0)} = 510e527f$

$f = H_5^{(0)} = 9b05688c$

$g = H_6^{(0)} = 1f83d9ab$

$h = H_7^{(0)} = 5be0cd19$

h) Proses kompresi 64 putaran:

- Menghitung T_1 dan T_2:

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$

$$T_2 = \sum_0^{\{256\}}(a) + Ma(a,b,c)$$

- Memperbarui variabel $a, b, c, \dots, h$:

$h = g$

$g = f$

$f = e$

$e = d + T_1$

$d = c$

$c = b$

$b = a$

$a = T_1 + T_2$

Putaran ke-1 ($t = 0$)

- Diketahui $a = 6a09e667,\ \ b = bb67ae85,\ \ c = 3c6ef372,$ $d = a54ff53a,\ e = 510e527f,\ f = 9b05688c,\ g = 1f83d9ab,\ h = 5be0cd19$,

  a dalam bentuk biner: 01101010000010011110011001100111

  e dalam bentuk biner: 01010001000011100101001001111111

  $\neg e$: 10101110111100011101011010000000 atau AEF1D680

- $\sum_0^{\{256\}}(a) = ROTR^2(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a)$

  $\sum_0^{\{256\}}(a) = 11011010100000100111100110011001$

  $\oplus\ 00110011001110110101000001001111$

  $\oplus\ 00100111100110011001110110101000$

  $\sum_0^{\{256\}}(a) = 11001110001000001011010001111110$

  $= ce20b47e$

- $\sum_1^{\{256\}}(e) = ROTR^6(e) \oplus ROTR^{11}(e) \oplus ROTR^{25}(e)$

  $= 11111101010001000011100101001001$

  $\oplus 01001111111010100010000111001010$

  $\oplus 10000111001010010011111110101000$

  $\sum_1^{\{256\}}(e) = 00110101100001110010011100101011$

  $= 3587272b$

- $Ma(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$

  $Ma(a,b,c) = \big((6a09e667) \wedge (bb67ae85)\big)$

  $\oplus\big((6a09e667) \wedge (3c6ef372)\big) \oplus ((bb67ae85) \wedge$

  $(3c6ef372))$

$$Ma(a,b,c) = 00111010011011111110011001100111$$
$$= 3a6fe667$$

- $Ch(e,f,g) = (e \wedge f) \oplus ((\neg e) \wedge g)$

  $Ch(e,f,g) = (510e527f \wedge 9b05688c) \oplus (aef1d680 \wedge 1f83d9ab)$

  $Ch(e,f,g) = 00011111100001011100100110001100$
  $= 1f85c98c$

- $T_1 = h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_0^{\{256\}} + W_0$

  $T_1 = 5be0cd19 + 3587272b + 1f85c98c + 428a2f98 + 4b52a950$

  $T_1 = 3eca36b8$

- $T_2 = \sum_0^{\{256\}}(a) + Maj(a,b,c)$

  $T_2 = ce20b47e + 3a6fe667$

  $T_2 = 08909ae5$

- Perbarui variabel

  $h = g = 1f83d9ab$

  $g = f = 9b05688c$

  $f = e = 510e527f$

  $e = d + T_1 = a54ff53a + 3eca36b8 = e41a2bf2$

  $d = c = 3c6ef372$

  $c = b = bb67ae85$

  $b = a = 6a09e667$

  $a = T_1 + T_2 = 3eca36b8 + 08909ae5 = 475ad19d$

Langkah ini diulang hingga putaran ke-64 ($t = 63$).

i) Penjumlahan *buffer* akhir: Setelah semua putaran selesai, *buffer* SHA-256 diperbarui dengan menjumlahkan nilai *buffer* sesudah pembaruan (a') dengan nilai awal *buffer* (a), seperti berikut:

$H_0^{(0)} = ca56e92c + 6a09e667 = 3460cf93$

$H_1^{(0)} = 5145929f + bb67ae85 = 0cad4124$

$H_2^{(0)} = 4aaccd0e + 3c6ef372 = 871bc080$

$H_3^{(0)} = 4c9a20f1 + a54ff53a = f1ea162b$

$H_4^{(0)} = e5e93cc1 + 510e527f = 36f78f40$

$H_5^{(0)} = 06909946 + 9b05688c = a19601d2$

$H_6^{(0)} = 5f56e4c8 + 1f83d9ab = 7edabe73$

$H_7^{(0)} = e1dcaefe + 5be0cd19 = 3dbd7c17$

Selanjutnya menggabungkan $H_0^{(0)} - H_7^{(0)}$ hingga diperoleh *message digest* (m), yaitu:

$m = 3460cf930cad4124871bc080f1ea162b36f78f40$

$a19601d27edabe733dbd7c17$

j) *Message digest* (m) yang diperoleh tersusun dari hasil penggabungan nilai akhir setiap *buffer* dalam format heksadesimal. Untuk memahami nilai numeriknya dalam sistem desimal, dilakukan proses konversi bilangan heksadesimal ke desimal berdasarkan Tabel ASCII, dengan langkah-langkah sebagai berikut:

- Menentukan posisi digit. Setiap digit dalam sistem heksadesimal memiliki bobot berdasarkan pangkat 16, di mana digit paling kanan memiliki posisi 0, digit berikutnya memiliki posisi 1, dan

seterusnya hingga digit paling kiri.

Secara umum, bilangan heksadesimal N dengan n digit dapat dinyatakan dalam bentuk:

$$N = \sum_{i=0}^{n} d_i \times 16^i$$

Dengan d_i merupakan nilai desimal dari digit heksadesimal pada posisi ke-i.

- Konversi digit heksadesimal ke desimal. Setiap digit dikonversi ke desimal sebelum digunakan dalam perhitungan lebih lanjut.
- Perhitungan konversi ke desimal. Setiap digit dikalikan dengan 16 pangkat posisinya dan dijumlahkan sebagai berikut:

  $N = (3 \times 16^{63}) + (4 \times 16^{62}) + (6 \times 16^{61}) + \cdots + (7 \times 16^{0})$

  $N = 236913180704895430612891178891397029410621583506624229136395979928261433415 91$

  Jadi, *message digest* (e) hasil representasi desimal dari bilangan heksadesimal adalah:

  $e = 236913180704895430612891178891397029410621583506624229136395979928261433415 91$

2. Mengenkripsi *message digest* menggunakan algoritma ECDSA

   a) Tahap penentuan titik kurva eliptik

   - Dipilih bilangan prima $p = 11$ sebagai modulus bidang hingga $GF(11)$.
   - Dipilih koefisien $a = 1$ dan $b = 6$, sehingga diperoleh persamaan kurva eliptik:

$$y^2 \equiv x^3 + x + 6 \ (mod\ 11)$$

- Ditentukan elemen-elemen dalam $GF(11)$, yaitu $\{0,1,2, \dots, 10\}$. Untuk setiap nilai x dalam $GF(11)$, dihitung $x^3 + x + 6 \ (mod\ 11)$, kemudian dicari nilai y yang memenuhi $y^2 \equiv x^3 + x + 6 \ (mod\ 11)$.
  - Untuk $x = 0 \ \rightarrow \ y^2 \equiv 6 \ (mod\ 11)$.
  - Untuk $x = 1 \ \rightarrow \ y^2 \equiv 8 \ (mod\ 11)$.
  - Untuk $x = 2 \ \rightarrow y^2 \equiv 16 \ mod\ 11 \equiv 5 \ (mod\ 11) \rightarrow y_1 = 4 \ dan \ y_2 = 7 \rightarrow P_1(2,4)$ dan $P_2(2,7)$.
  - Untuk $x = 3 \ \rightarrow y^2 \equiv 36 \ mod\ 11 \equiv 3 \ (mod\ 11) \rightarrow y_1 = 5 \ dan \ y_2 = 6 \rightarrow P_1(3,5)$ dan $P_2(3,6)$.

    ⋮
  - Untuk $x = 10 \ \rightarrow y^2 \equiv 1.016 \ mod\ 11 \equiv 4 (mod\ 11) \rightarrow y_1 = 2 \ dan \ y_2 = 9 \rightarrow P_1(10,2)$ dan $P_2(10,9)$.
- Dengan demikian, kurva eliptik $y^2 \equiv x^3 + x + 6 \ (mod\ 11)$ memiliki 12 titik, yaitu: $(2,4), (2,7), (3,5), (3,6), (5,2), (5,9), (7,2), (7,9), (8,3), (8,8), (10,2), (10,9)$. Jika titik di tak hingga O dimasukkan, maka himpunan titik-titik ini membentuk grup dengan jumlah elemen $n = 13$

b) Tahap penentuan kunci

- Dipilih $d_A = 3$ sebagai sebuah bilangan bulat random yang nilainya diantara $[1, n-1]$ dan $G = (2,4)$ yang merupakan salah satu titik pada kurva eliptik $y^2 \equiv x^3 + x + 6 \ (mod\ 11)$
- Menghitung Q_A dengan rumus:

$Q_A = d_A \cdot G = (x_1, y_1)$

$Q_A = 3 \cdot (2,4) = (8,8)$

- Sehingga di peroleh kunci rahasia $(d_A) = 3$, dan kunci publik $(Q_A) = (8,8)$

c) Tahap Penandatanganan

- Dipilih $k = 5$ sebagai sebuah bilangan bulat random yang nilainya diantara $[1, n-1]$ dengan $n = 13$
- Menghitung R dengan rumus:

  $R = k \cdot G = (x_1, y_1)$

  $R = 5 \cdot (2,4) = (3,5)$

  Kemudian menghitung r dengan rumus:

  $r = x_1 \; mod \; n$

  $r = 3 \; mod \; 13 = 3$
- Menghitung $k^{-1} mod \; 13$, untuk mencari k^{-1} dapat dilakukan dengan menggunakan algoritma *Euclidean Extended* yaitu $k \cdot k^{-1} \equiv 1 \; mod \; 13$, sehingga mendapatkan hasil $k^{-1} \equiv 8 \; mod \; 13$
- Menghitung $e = Hash(m)$. Dalam proses enkripsi, nilai dari *message digest* diubah dalam bentuk desimal, sehingga menghasilkan

  $e = 236913180704895430612891178891397029410621$

  $58350662422913639597992826143341591$
- Setelah nilai dari *message digest* diubah ke dalam bentuk desimal, dilakukan perhitungan untuk memperoleh nilai s yang merupakan bagian dari pasangan tanda tangan digital (r, s), dengan rumus:

$s = k^{-1}\{e + d_A \cdot r\}\, mod\ n$

$s = 8\,\{(2369131807048954306128911788913970294106215835066242291363959799282614334159 1) +$

$3 \cdot 3\}\, mod\ 13$

$s = 8\,\{(2369131807048954306128911788913970294106215835066242291363959799282614334159 1) +$

$9)\, mod\ 13\}\ \ mod\ 13$

$s = 8 \cdot 1\, mod\ 13$

$s = 8\, mod\ 13 = 8$

Jadi, tanda tangan untuk pesan "KRIPTOGRAFI" menghasilkan titik $(3, 8)$ sebagai cipherteks, yang menunjukkan integritas dan keaslian pesan tersebut dalam konteks kriptografi

3. Tahap Verifikasi

a) Memverifikasi bahwa r dan s adalah bilangan bulat yang antara $[1, n-1]$

Sudah diketahui bahwa

$e = 2369131807048954306128911788913970294106215835066242291363959799282614334159 1$

b) Menghitung w menggunakan rumus:

$w = s^{-1}\, mod\ n$

$w = 8^{-1}\, mod\ 13 = 5$

c) Menghitung u_1 dan u_2 dengan rumus:

$u_1 = e \cdot w\, mod\ n$

$u_1 = (2369131807048954306128911788913970294106215835066242291363959799282614334159 1$

$158350662422913639597992826143341591)\ 5\ mod\ 13$

$u_1 = 12$

$u_2 = r \cdot w\ mod\ n$

$u_2 = 3 \cdot 5\ mod\ 13$

$u_2 = 2$

d) Menghitung titik (x_1, y_1) dengan menggunakan rumus:

$u_1 \cdot G + u_2 \cdot Q_A = (x_1, y_1)$

$12 \cdot (2,4) + 2 \cdot (8,8) = (x_1, y_1)$

$(2,7) + (7,2) = (3,5)$

$(x_1, y_1) = (3,5)$

e) Menghitung v dan z dengan menggunakan rumus berikut:

$v = x_1\ mod\ n$

$v = 3\ mod\ 13 = 3$

$z = y_1\ mod\ n$

$z = 5\ mod\ 13 = 5$

Menerima tanda tangan jika dan hanya jika $v = r$, dan terbukti bahwa v dan r sama-sama bernilai 3, jadi dapat dikatakan pesan tersebut valid.

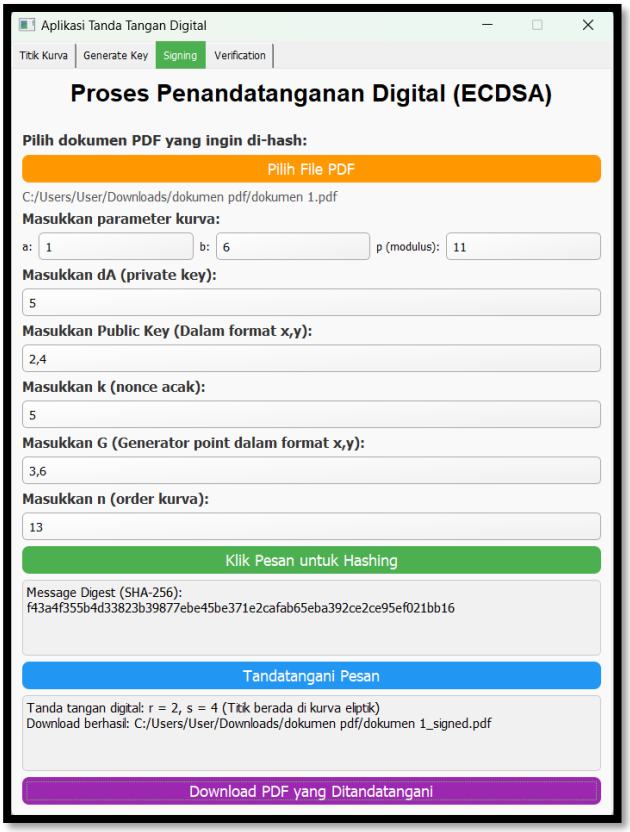
4.2 Pengujian dan Analisis Data

4.2.1 Proses Penandatanganan Dokumen

Dalam bab ini, penelitian ini menggunakan bahasa pemrograman *Python* yaitu PyQt5 untuk menguji pembentukan tanda tangan digital. Proses ini melibatkan penggunaan fungsi *hash* SHA-256 serta metode kriptografi *Elliptic Curve Digital Signature* (ECDSA) pada tiga puluh dokumen yang disimpan dalam

format PDF. Pengujian ini bertujuan untuk mengevaluasi keakuratan dan keandalan algoritma dalam menghasilkan serta memverifikasi tanda tangan digital, sehingga dapat memastikan integritas, autentikasi, dan keamanan dokumen yang ditandatangani. Selain itu, pengujian ini juga bertujuan untuk memastikan bahwa algoritma dapat diterapkan secara konsisten pada berbagai dokumen dengan parameter kurva eliptik yang telah ditentukan.

Pengujian pertama dilakukan pada dokumen yang bernama "dokumen 1.pdf", yang memiliki dua halaman. Dalam pengujian ini, digunakan parameter kurva eliptik dengan nilai $a = 1,\ b = 6$, dan $modulus\ p = 11$. Kunci privat yang digunakan adalah $d_A = 5$ dengan kunci publik $Q_A = (2, 4)$. Selain itu, bilangan acak yang dipilih adalah $k = 5$, titik generator $G = (3, 6)$, serta orde kurva $n = 13$. Hasil dari pengujian ini akan dipaparkan sebagai berikut:



Gambar 4.1 Hasil Tanda Tangan Digital "dokumen 1.pdf"

Message digest yang diperoleh, sebagaimana ditunjukkan pada Gambar 4.1, adalah:

$$f43a4f355b4d33823b39877ebe45be371e2cafab65eba392c$$

$$e2ce95ef021bb16$$

Message digest tersebut kemudian dikonversi ke dalam format desimal, menghasilkan nilai

$$1104673588605278921436128488197820363973266865149 2$$

$$6964814151736311168849001238$$

Selanjutnya, kunci pribadi (d_A) akan digunakan untuk mengenkripsi representasi desimal dari *message digest*. Pada pengujian "dokumen 1", nilai d_A ditetapkan sebesar 5, yang dipilih sebagai bilangan bulat acak dalam rentang $[1, n-1]$, dengan $n = 13$, yaitu jumlah titik pada kurva eliptik yang telah ditentukan. Berdasarkan nilai d_A tersebut, kunci publik Q_A dihitung sebagai perkalian titik G pada kurva dengan d_A, yaitu:

$$Q_A = d_A \cdot G = 5 \cdot (3,6) = (2,4)$$

Pada enkripsi ini, nilai k juga ditentukan sebagai bilangan bulat acak yang memiliki invers dalam modulo n dan berada dalam rentang yang sama, dimana k untuk pengujian ini adalah 5. Titik G yang digunakan adalah $(3,6)$. Langkah-langkah perhitungan sebagai berikut:

1. Menghitung titik $R = k \cdot G$

   $R = 5 \cdot (3,6) = (2,4)$

2. Menentukan r sebagai $r = x_1 \; mod \; n$

   $r = 2 \; mod \; 13 = 2$

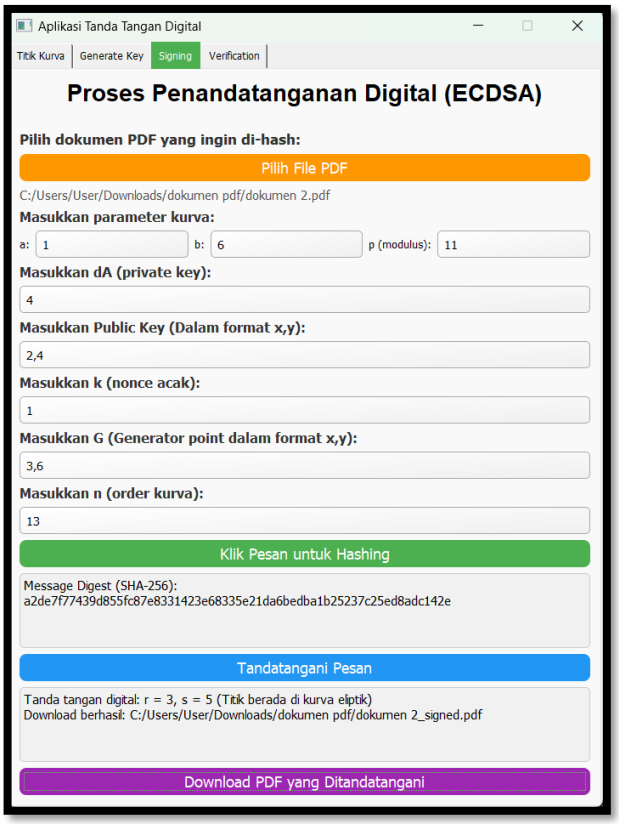
3. Menghitung $s = k^{-1}\{e + d_A \cdot r\} \; mod \; n$

$$s = 8 \begin{Bmatrix} 110467358860527892143612848 81 \\ 97820363973266865149269648141 51 \\ 736311168849001238 + 5 \cdot 2 \end{Bmatrix} mod\ 13$$

$$s = 4$$

Dengan demikian, pasangan tanda tangan digital yang dihasilkan pada pengujian “dokumen 1” adalah $(r, s) = (2, 4)$.

Pengujian kedua dilakukan pada dokumen yang bernama “dokumen 2.pdf”, yang memiliki satu halaman. Dalam pengujian ini, digunakan parameter kurva eliptik dengan nilai $a = 1$, $b = 6$, dan $modulus\ p = 11$. Kunci privat yang digunakan adalah $d_A = 4$ dengan kunci publik $Q_A = (2, 4)$. Selain itu, bilangan acak yang dipilih adalah $k = 1$, titik generator $G = (3, 6)$, serta orde kurva $n = 13$. Hasil dari pengujian ini akan dipaparkan sebagai berikut:



Gambar 4.2 Hasil Tanda Tangan Digital "dokumen 2.pdf"

Message digest yang diperoleh, sebagaimana ditunjukkan pada Gambar 4.2, adalah:

a2de7f77439d855fc87e8331423e68335e21da6bedb

a1b25237c25ed8adc142e

Message digest tersebut kemudian dikonversi ke dalam format desimal, menghasilkan nilai

736678012560103759498419891467667972711739300 73

291671625553355254678557561902

Selanjutnya, kunci pribadi (d_A) akan digunakan untuk mengenkripsi representasi desimal dari *message digest*. Pada pengujian "dokumen 2", nilai d_A ditetapkan sebesar 4, yang dipilih sebagai bilangan bulat acak dalam rentang $[1, n-1]$, dengan $n = 13$, yaitu jumlah titik pada kurva eliptik yang telah ditentukan. Berdasarkan nilai d_A tersebut, kunci publik Q_A dihitung sebagai perkalian titik G pada kurva dengan d_A, yaitu:

$$Q_A = d_A \cdot G = 4 \cdot (3,6) = (2,4)$$

Pada enkripsi ini, nilai k juga ditentukan sebagai bilangan bulat acak yang memiliki invers dalam modulo n dan berada dalam rentang yang sama, dimana k untuk pengujian ini adalah 1. Titik G yang digunakan adalah $(3,6)$. Langkah-langkah perhitungan sebagai berikut:

1. Menghitung titik $R = k \cdot G$

   $R = 1 \cdot (3,6) = (3,6)$

2. Menentukan r sebagai $r = x_1 \; mod \; n$

   $r = 3 \; mod \; 13 = 3$

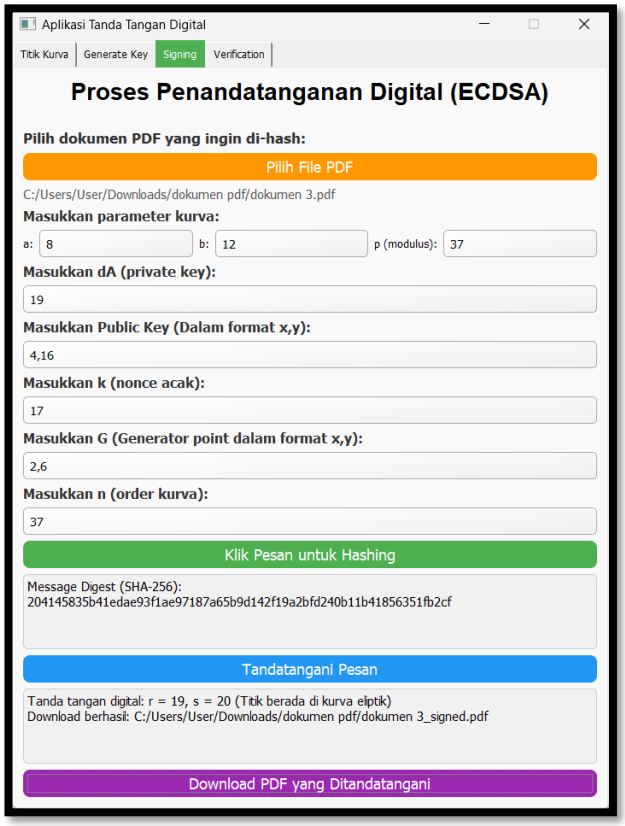
3. Menghitung $s = k^{-1}\{e + d_A \cdot r\} \; mod \; n$

$$s = 1 \left\{ \begin{pmatrix} (736678012560103759498419891467 \\ 667972711739300732916716255533552 \\ 54678557561902) + 5 \cdot 3 \end{pmatrix} \right\} mod\ 13$$

$$s = 5$$

Dengan demikian, pasangan tanda tangan digital yang dihasilkan pada pengujian “dokumen 2” adalah $(r, s) = (3, 5)$.

Pengujian ketiga dilakukan pada dokumen yang bernama “dokumen 3.pdf”, yang memiliki satu halaman. Dalam pengujian ini, digunakan parameter kurva eliptik dengan nilai $a = 8,\ b = 12$, dan $modulus\ p = 37$. Kunci privat yang digunakan adalah $d_A = 19$ dengan kunci publik $Q_A = (4, 16)$. Selain itu, bilangan acak yang dipilih adalah $k = 17$, titik generator $G = (2, 6)$, serta orde kurva $n = 37$. Hasil dari pengujian ini akan dipaparkan sebagai berikut:



Gambar 4.3 Hasil Tanda Tangan Digital "dokumen 3.pdf"

Message digest yang diperoleh, sebagaimana ditunjukkan pada Gambar 4.3, adalah:

204145835b41edae93f1ae97187a65b9d142f19a

2bfd240b11b41856351fb2cf

Message digest tersebut kemudian dikonversi ke dalam format desimal, menghasilkan nilai

145893359757366012860593765369973557018682327

292813051037106766852214688119 83

Selanjutnya, kunci pribadi (d_A) akan digunakan untuk mengenkripsi representasi desimal dari *message digest*. Pada pengujian "dokumen 3", nilai d_A ditetapkan sebesar 19, yang dipilih sebagai bilangan bulat acak dalam rentang $[1, n-1]$, dengan $n = 37$, yaitu jumlah titik pada kurva eliptik yang telah ditentukan. Berdasarkan nilai d_A tersebut, kunci publik Q_A dihitung sebagai perkalian titik G pada kurva dengan d_A, yaitu:

$$Q_A = d_A \cdot G = 19 \cdot (2, 6) = (4, 16)$$

Pada enkripsi ini, nilai k juga ditentukan sebagai bilangan bulat acak yang memiliki invers dalam modulo n dan berada dalam rentang yang sama, dimana k untuk pengujian ini adalah 17. Titik G yang digunakan adalah $(2, 6)$. Langkah-langkah perhitungan sebagai berikut:

1. Menghitung titik $R = k \cdot G$

   $R = 17 \cdot (2, 6) = (19, 17)$

2. Menentukan r sebagai $r = x_1 \, mod \, n$

   $r = 19 \, mod \, 37 = 19$

3. Menghitung $s = k^{-1}\{e + d_A \cdot r\} \, mod \, n$

$$s = 24 \begin{Bmatrix} (145893359757366012860593 76 \\ 536997355701868232729281 3 \\ 05103710676685221468811983) + 19 \cdot 19 \end{Bmatrix} mod\ 37$$

$$s = 20$$

Dengan demikian, pasangan tanda tangan digital yang dihasilkan pada pengujian "dokumen 3" adalah $(r, s) = (19, 20)$.

Hasil pengujian tanda tangan digital terhadap tiga puluh dokumen elektronik yang diuji tercantum dalam Tabel 4.2. Dari total tiga puluh dokumen yang diuji, sebanyak dua puluh lima dokumen tidak mengalami perubahan apapun, baik itu pada isi dokumen. Sementara itu, lima dokumen lainnya mengalami modifikasi pada elemen tertentu untuk menguji ketahanan sistem dalam mendeteksi perubahan terhadap integritas tanda tangan digital.

Berikut Tabel 4.1 yang menunjukkan jenis perubahan yang diterapkan pada lima dokumen tersebut:

Tabel 4.1 Perubahan yang Diterapkan pada Dokumen

No	Dokumen	Jenis Perubahan yang diterapkan
26	dokumen 25	Perubahan tanda tangan digital
27	dokumen 25	Perubahan kunci publik
28	dokumen 20	Penambahan 1 karakter
29	dokumen 10	Pertukaran posisi kata (jumlah karakter dan kata tetap)
30	dokumen 16	Perubahan 2 karakter (jumlah karakter dan kata tetap)

Setelah proses penandatanganan digital selesai, seluruh dokumen yang telah diuji, baik yang asli maupun yang telah dimodifikasi, dianalisis lebih lanjut. Analisis ini mencakup karakteristik setiap dokumen seperti jumlah kata dan karakter, nama dokumen, serta hasil *hashing* dalam bentuk *message digest*, kunci privat yang digunakan, dan tanda tangan digital yang dihasilkan. Hasil pengujian terhadap ketiga puluh dokumen tersebut dapat dilihat pada Tabel 4.2

Tabel 4.2 Hasil Tanda Tangan Digital terhadap 30 Dokumen PDF

No	Jumlah		Nama Dokumen	*Message Digest*	Kunci Pribadi	Tanda Tangan Digital (r, s)
	Kt	Kr				
1	257	1749	dokumen 1.pdf	f43a4f355b4d33823b39877ebe45be371e2cafab65eba392ce2ce95ef021bb16	5	(2, 4)
2	85	726	dokumen 2.pdf	a2de7f77439d855fc87e8331423e68335e21da6bedba1b25237c25ed8adc142e	4	(3, 5)
3	157	1398	dokumen 3.pdf	204145835b41edae93f1ae97187a65b9d142f19a2bfd240b11b41856351fb2cf	19	(19, 20)
4	204	1374	dokumen 4.pdf	6c50bceda74574dc7a1d66ecfde06fab10711214e0f33a8529551614cdf74e2e	19	(33, 29)
5	151	1194	dokumen 5.pdf	5e88da288a02369d195d9e4c1a1df74f85196ee086000ef9b9f6fe8c5fe6cf1a	19	(9, 31)
6	164	1220	dokumen 6.pdf	ba25abe53b7011a18954367711cbd8de9ef31f6e03ebc4541b26d16a0850882b	19	(4, 21)
7	175	1391	dokumen 7.pdf	151c25c4a8a87a1f43275b0ea787a3feaee6e66cf2d6490f758aa53c89834209	19	(4, 16)
8	179	1320	dokumen 8.pdf	6cba266ef64b78afeadb23f9c98486ea7b134b0b6897661ad7235071601bb8bf	19	(1, 24)
9	141	1113	dokumen 9.pdf	929ff1b762a69013184b14d574826c5b615581ee24bffe2c091406026bcb4733	19	(1, 13)
10	184	1298	dokumen 10.pdf	4104e1f0fa0c52f796d86b1776c9fb491cc62d351241ed7be466bedbbab7b617	19	(19, 20)
11	191	944	dokumen 11.pdf	0c38e63027d1defc5d6bbee5c5bb1d4e4be72e43d740d55d7c6c1072196973a9	2	(15, 6)
12	71	536	dokumen 12.pdf	e750c156a481e7442d3919f1f5d303e57d2ca1d9940f7b47f719aaeecbf1299e	5	(1, 33)
13	145	1114	dokumen 13.pdf	26c215a1d8235b2427781819b71e0b7efb7810c6149ce35e84885c969b281486	5	(26, 21)
14	185	1309	dokumen 14.pdf	10ade92c5e0579873344fa748a44f795ad875a31370b83850f52c81ea6ceb964	3	(45, 19)
15	137	1111	dokumen 15.pdf	c1d154d9ee8504efb0a8e2f630f8e46b89c1ee9f59b87c24307e8bdfba2ebd21	3	(22, 45)
16	174	1205	dokumen 16.pdf	c59017c29283753f408a1b1f0ecae7c5ae60d61defa6cebe40d804dc8641d769	2	(47, 6)
17	153	1230	dokumen 17.pdf	55281a2f9cc2c1cca4424d08b6151829c280d043f89ef4d4807e2a72a78bc5c9	3	(94, 89)
18	171	1298	dokumen 18.pdf	42336cba20237e3be958cefbd43725d36850b52a51232d630f5e214610ba8792	3	(60, 16)
19	153	1144	dokumen 19.pdf	f2e46a2e48d63b59b6cb1c9b33e787cf13555ac8eedd4c4133ebc66999b3ccda	4	(43, 25)

No	Jumlah		Nama Dokumen	*Message Digest*	Kunci Pribadi	Tanda Tangan Digital (r, s)
	Kt	Kr				
20	173	1268	dokumen 20.pdf	efeadee469e162d6e27f2b45c1e0535ad1f8638ca6c05535b83262ecf6708b0d	2	(12, 58)
21	146	1165	dokumen 21.pdf	eef845cad6b62098f365b0492f8ed9ba6c4847deead65c56807ef2b08f05adfa	12	(10, 4)
22	790	5614	dokumen 22.pdf	40e21e2fdb0d9e3d8650002487c1a97d8218f690c28503907a6c174e64c196be	8	(10, 4)
23	537	4267	dokumen 23.pdf	472b837c312d7c30ea988bbfd0d9005ee0816b7342d225b750ce9a57aa26ce48	5	(69, 36)
24	556	4424	dokumen 24.pdf	0f4f138f9448995cdf1ad6033c8cba6acef0052b3e2bac8b2649c039503bc2ae	5	(77, 24)
25	601	4410	dokumen 25.pdf	5366ea922778b52c4d1afc9bc90c3dabc283479db07062c5f6b94d4a5843c81c	2	(26, 8)
26	601	4410	dokumen 26.pdf	5366ea922778b52c4d1afc9bc90c3dabc283479db07062c5f6b94d4a5843c81c	2	(27, 8)
27	601	4410	dokumen 27.pdf	5366ea922778b52c4d1afc9bc90c3dabc283479db07062c5f6b94d4a5843c81c	2	(26, 8)
28	174	1268	dokumen 28.pdf	c0dbb98211478cedfd84c6915a190ecbf1e37079a13657e0a5a855e1579f1414	2	(12, 58)
29	283	2115	dokumen 29.pdf	c8b62583315f5b10baa87a41c5acd3246f733c06b56e1bc015dda7ef8d03d128	19	(19, 20)
30	137	1131	dokumen 30.pdf	2b4b772ffd9cbc209f58eb3052636fbf04fc7183e06df99b07dd56862da50970	2	(47, 6)

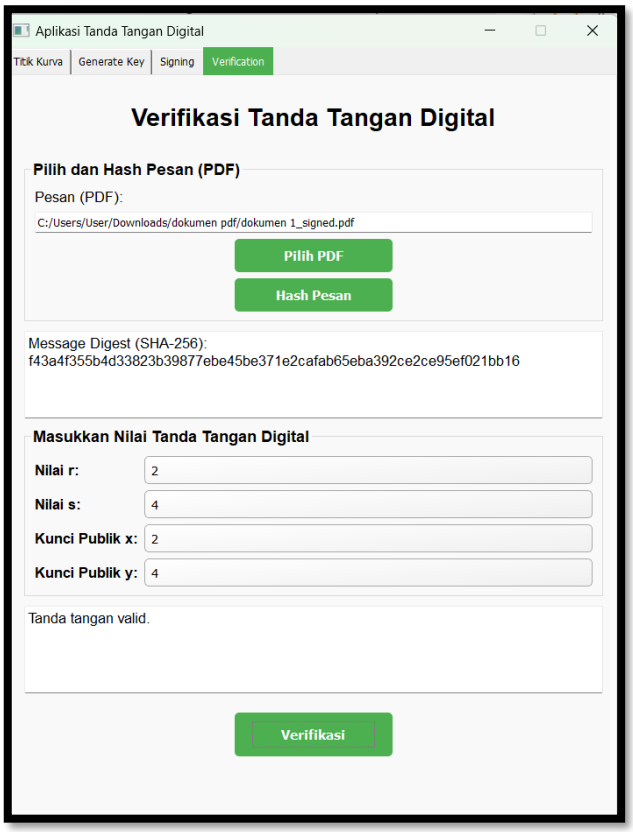
Keterangan :

- Kt : Kata
- Kr : Karakter

4.2.2 Proses Verifikasi Dokumen

Bab ini membahas proses verifikasi tanda tangan digital menggunakan bahasa pemrograman *Python* dan PyQt5 pada tiga puluh dokumen PDF. Verifikasi dilakukan dengan menerapkan fungsi *hash* SHA-256 dan metode dekripsi ECDSA. Dari total dokumen yang diuji, sebanyak dua puluh lima dokumen tetap utuh tanpa perubahan, sementara lima dokumen lainnya mengalami modifikasi untuk menguji ketahanan sistem dalam mendeteksi perubahan pada elemen-elemen krusial, seperti isi dokumen, tanda tangan digital, dan kunci publik. Jenis perubahan yang diterapkan pada lima dokumen tersebut tercantum dalam Tabel 4.1.

Pengujian awal dilakukan pada “dokumen 1.pdf”, yang memiliki dua halaman. Hasil dari pengujian ini dipaparkan sebagai berikut:



Gambar 4.4 Hasil Verifikasi "dokumen 1.pdf"

Berdasarkan Gambar 4.4, tanda tangan digital yang tertera yaitu pasangan $(r, s) = (2, 4)$, didekripsikan menggunakan kunci publik. Kunci publik Q_A, yang diperoleh dari proses penandatanganan dokumen pertama bernama "dokumen 1.pdf" pada sub bab 4.2.1, adalah $Q_A = (2, 4)$. Tanda tangan digital didekripsikan dengan menggunakan rumus verifikasi berikut dan langkah-langkah perhitungan:

1. Menghitung $w = s^{-1}\ mod\ n$

   $w = 4^{-1}\ mod\ 13 = 10$

2. Menghitung $u_1 = e \cdot w\ mod\ n$ dan $u_2 = r \cdot w\ mod\ n$

   $$u_1 = \begin{pmatrix} 1104673588605278921436128 4881 \\ 9782036397326686514926964814151 \\ 736311168849001238 \end{pmatrix} \cdot 10\ mod\ 13 = 9$$

   $u_2 = 2 \cdot 10\ mod\ 13 = 7$

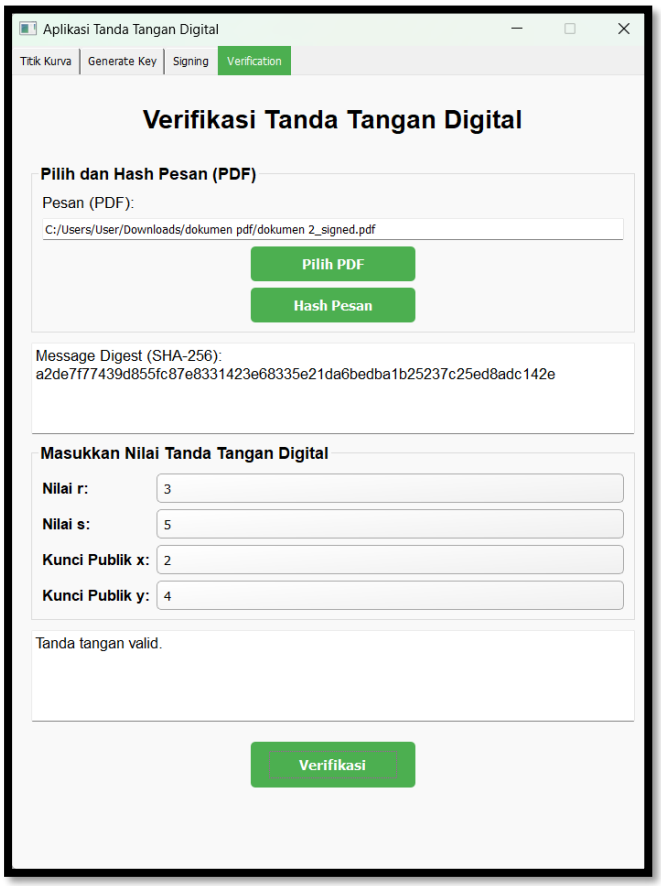
3. Menentukan $u_1 \cdot G + u_2 \cdot Q_A = (x_1, y_1)$ dan menghitung $v = x_1\ mod\ n$ dan $z = x_2\ mod\ n$

   $9 \cdot (3, 6) + 7 \cdot (2, 4) = (2, 4) \rightarrow v = 2\ mod\ 13 = 2$

Berdasarkan perhitungan yang telah dilakukan, hasil verifikasi menunjukkan bahwa nilai $v = r$ yang bernilai 2. Hal ini mengindikasikan bahwa tanda tangan digital pada dokumen tersebut terbukti autentik dan valid. Dengan demikian, proses verifikasi berhasil, yang menunjukkan bahwa dokumen tidak mengalami perubahan sejak proses penandatanganan.

Kesesuaian antara nilai v dan r menunjukkan bahwa parameter dan algoritma ECDSA telah diterapkan secara tepat, sehingga integritas dan keamanan dokumen terjamin. Hasil verifikasi ini mencerminkan konsistensi sistem dalam mendeteksi adanya modifikasi, yang merupakan indikator keandalan metode verifikasi dalam lingkungan operasional nyata.

Pengujian kedua dilakukan pada “dokumen 2.pdf”, yang memiliki satu halaman. Hasil dari pengujian ini dipaparkan sebagai berikut:



Gambar 4.5 Hasil Verifikasi "dokumen 2.pdf"

Berdasarkan Gambar 4.5, tanda tangan digital yang tertera yaitu pasangan $(r, s) = (3, 5)$, didekripsikan menggunakan kunci publik. Kunci publik Q_A, yang diperoleh dari proses penandatanganan dokumen kedua bernama “dokumen 2.pdf” pada sub bab 4.2.1, adalah $Q_A = (2, 4)$. Tanda tangan digital didekripsikan dengan menggunakan rumus verifikasi berikut dan langkah-langkah perhitungan:

1. Menghitung $w = s^{-1}\ mod\ n$

   $w = 5^{-1}\ mod\ 13 = 8$

2. Menghitung $u_1 = e \cdot w\ mod\ n$ dan $u_2 = r \cdot w\ mod\ n$

$$u_1 = \begin{pmatrix} 736678012560103759498419891467 \\ 667972711739300732916716255533552 \\ 54678557561902 \end{pmatrix} \cdot 8 \; mod \; 13 = 9$$

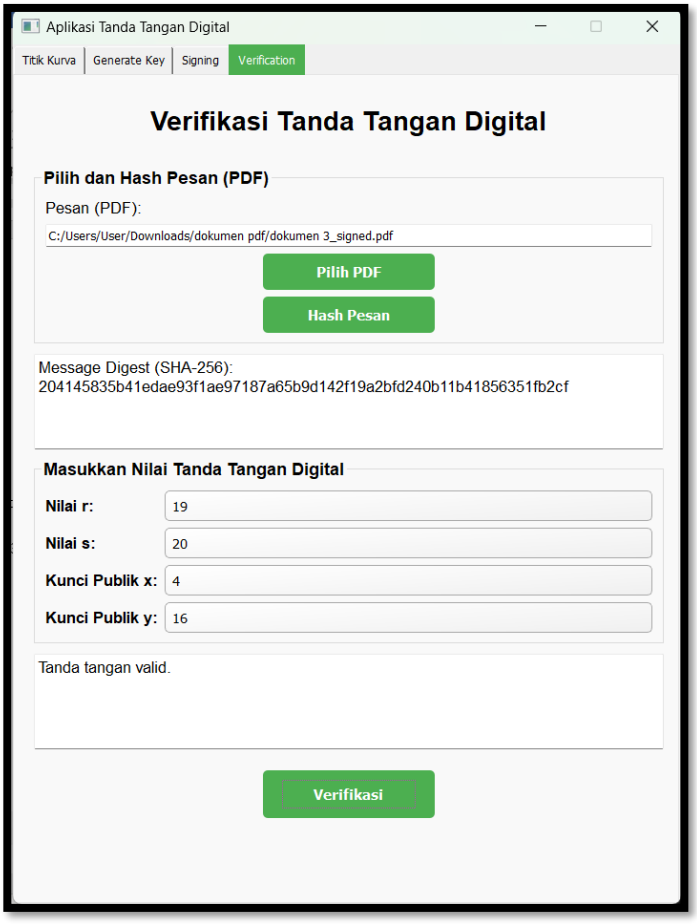
$$u_2 = 3 \cdot 8 \; mod \; 13 = 11$$

3. Menentukan $u_1 \cdot G + u_2 \cdot Q_A = (x_1, y_1)$ dan menghitung $v = x_1 \; mod \; n$ dan $z = x_2 \; mod \; n$

$$9 \cdot (3,6) + 11 \cdot (2,4) = (3,7) \rightarrow v = 3 \; mod \; 13 = 3$$

Hasil verifikasi menunjukkan bahwa nilai $v = r$ yang bernilai 3, yang mengindikasikan bahwa tanda tangan digital valid dan dokumen tetap autentik sejak ditandatangani.

Pengujian ketiga dilakukan pada “dokumen 3.pdf”, yang memiliki satu halaman. Hasil dari pengujian ini dipaparkan sebagai berikut:



Gambar 4.6 Hasil Verifikasi "dokumen 3.pdf"

Berdasarkan Gambar 4.6, tanda tangan digital yang tertera yaitu pasangan $(r, s) = (19, 20)$, didekripsikan menggunakan kunci publik. Kunci publik Q_A, yang diperoleh dari proses penandatanganan dokumen kedua bernama "dokumen 3.pdf" pada sub bab 4.2.1, adalah $Q_A = (4, 16)$. Tanda tangan digital didekripsikan dengan menggunakan rumus verifikasi berikut dan langkah-langkah perhitungan:

1. Menghitung $w = s^{-1}\ mod\ n$

   $w = 20^{-1}\ mod\ 37 = 13$

2. Menghitung $u_1 = e \cdot w\ mod\ n$ dan $u_2 = r \cdot w\ mod\ n$

   $$u_1 = \begin{pmatrix} 145893359757366012860593 76 \\ 536997355701868232729 2813 \\ 051037106766852214688119 83 \end{pmatrix} \cdot 13\ mod\ 37 = 23$$

   $u_2 = 19 \cdot 13\ mod\ 37 = 25$

3. Menentukan $u_1 \cdot G + u_2 \cdot Q_A = (x_1, y_1)$ dan menghitung $v = x_1\ mod\ n$ dan $z = x_2\ mod\ n$

   $23 \cdot (2, 6) + 25 \cdot (4, 16) = (19, 17) \rightarrow v = 19\ mod\ 37 = 19$

Berdasarkan perhitungan yang telah dilakukan, hasil verifikasi menunjukkan bahwa nilai $v = r$ yang bernilai 19. Hal ini mengindikasikan bahwa tanda tangan digital pada dokumen tersebut terbukti autentik dan valid. Dengan demikian, proses verifikasi berhasil, yang menunjukkan bahwa dokumen tidak mengalami perubahan sejak proses penandatanganan.

Hasil verifikasi tanda tangan digital terhadap tiga puluh dokumen elektronik yang diuji tercantum dalam Tabel 4.3

Tabel 4.3 Hasil Verifikasi Tanda Tangan Digital terhadap 30 Dokumen PDF

No	Nama Dokumen	*Message Digest*	Kunci Publik	Tanda Tangan Digital (r, s)	Dekripsi Tanda Tangan Digital (v, z)	Status
1	dokumen 1.pdf	f43a4f355b4d33823b39877ebe45be371e2cafab65eba392ce2ce95ef021bb16	(2, 4)	(2, 4)	(2, 4)	Valid
2	dokumen 2.pdf	a2de7f77439d855fc87e8331423e68335e21da6bedba1b25237c25ed8adc142e	(2, 4)	(3, 5)	(3, 5)	Valid
3	dokumen 3.pdf	204145835b41edae93f1ae97187a65b9d142f19a2bfd240b11b41856351fb2cf	(4, 16)	(19, 20)	(19, 17)	Valid
4	dokumen 4.pdf	6c50bceda74574dc7a1d66ecfde06fab10711214e0f33a8529551614cdf74e2e	(4, 16)	(33, 29)	(33, 29)	Valid
5	dokumen 5.pdf	5e88da288a02369d195d9e4c1a1df74f85196ee086000ef9b9f6fe8c5fe6cf1a	(4, 16)	(9, 31)	(9, 31)	Valid
6	dokumen 6.pdf	ba25abe53b7011a18954367711cbd8de9ef31f6e03ebc4541b26d16a0850882b	(4, 16)	(4, 21)	(4, 21)	Valid
7	dokumen 7.pdf	151c25c4a8a87a1f43275b0ea787a3feaee6e66cf2d6490f758aa53c89834209	(4, 16)	(4, 16)	(4, 16)	Valid
8	dokumen 8.pdf	6cba266ef64b78afeadb23f9c98486ea7b134b0b6897661ad7235071601bb8bf	(4, 16)	(1, 24)	(1, 24)	Valid
9	dokumen 9.pdf	929ff1b762a69013184b14d574826c5b615581ee24bffe2c091406026bcb4733	(4, 16)	(1, 13)	(1, 13)	Valid
10	dokumen 10.pdf	4104e1f0fa0c52f796d86b1776c9fb491cc62d351241ed7be466bedbbab7b617	(4, 16)	(19, 20)	(19, 20)	Valid

No	Nama Dokumen	*Message Digest*	Kunci Publik	Tanda Tangan Digital (r, s)	Dekripsi Tanda Tangan Digital (v, z)	Status
11	dokumen 11.pdf	0c38e63027d1defc5d6bbee5c5bb1d4e4be72e43d740d55d7c6c1072196973a9	(75, 87)	(15, 6)	(15, 6)	Valid
12	dokumen 12.pdf	e750c156a481e7442d3919f1f5d303e57d2ca1d9940f7b47f719aaeecbf1299e	(7, 77)	(1, 33)	(1, 17)	Valid
13	dokumen 13.pdf	26c215a1d8235b2427781819b71e0b7efb7810c6149ce35e84885c969b281486	(7, 77)	(26, 21)	(26, 15)	Valid
14	dokumen 14.pdf	10ade92c5e0579873344fa748a44f795ad875a31370b83850f52c81ea6ceb964	(47, 9)	(45, 19)	(45, 19)	Valid
15	dokumen 15.pdf	c1d154d9ee8504efb0a8e2f630f8e46b89c1ee9f59b87c24307e8bdfba2ebd21	(47, 9)	(22, 45)	(22, 44)	Valid
16	dokumen 16.pdf	c59017c29283753f408a1b1f0ecae7c5ae60d61defa6cebe40d804dc8641d769	(30, 41)	(47, 6)	(47, 91)	Valid
17	dokumen 17.pdf	55281a2f9cc2c1cca4424d08b6151829c280d043f89ef4d4807e2a72a78bc5c9	(84, 68)	(94, 89)	(94, 8)	Valid
18	dokumen 18.pdf	42336cba20237e3be958cefbd43725d36850b52a51232d630f5e214610ba8792	(84, 68)	(60, 16)	(60, 81)	Valid
19	dokumen 19.pdf	f2e46a2e48d63b59b6cb1c9b33e787cf13555ac8eedd4c4133ebc66999b3ccda	(59, 69)	(43, 25)	(43, 72)	Valid
20	dokumen 20.pdf	efeadee469e162d6e27f2b45c1e0535ad1f8638ca6c05535b83262ecf6708b0d	(23, 80)	(12, 58)	(12, 58)	Valid
21	dokumen 21.pdf	eef845cad6b62098f365b0492f8ed9ba6c4847deead65c56807ef2b08f05adfa	(10, 4)	(10, 4)	(10, 2)	Valid

No	Nama Dokumen	*Message Digest*	Kunci Publik	Tanda Tangan Digital (r, s)	Dekripsi Tanda Tangan Digital (v, z)	Status
22	dokumen 22.pdf	40e21e2fdb0d9e3d8650002487c1a97d8218f690c28503907a6c174e64c196be	(64, 68)	(10, 4)	(10, 79)	Valid
23	dokumen 23.pdf	472b837c312d7c30ea988bbfd0d9005ee0816b7342d225b750ce9a57aa26ce48	(31, 66)	(69, 36)	(69, 36)	Valid
24	dokumen 24.pdf	0f4f138f9448995cdf1ad6033c8cba6acef0052b3e2bac8b2649c039503bc2ae	(31, 66)	(77, 24)	(77, 24)	Valid
25	dokumen 25.pdf	5366ea922778b52c4d1afc9bc90c3dabc283479db07062c5f6b94d4a5843c81c	(1, 21)	(26, 8)	(26, 8)	Valid
26	dokumen 26.pdf	5366ea922778b52c4d1afc9bc90c3dabc283479db07062c5f6b94d4a5843c81c	(1, 21)	(27, 8)	(56, 50)	Tidak Valid
27	dokumen 27.pdf	5366ea922778b52c4d1afc9bc90c3dabc283479db07062c5f6b94d4a5843c81c	(3, 21)	(26, 8)	Error	Error
28	dokumen 28.pdf	c0dbb98211478cedfd84c6915a190ecbf1e37079a13657e0a5a855e1579f1414	(23, 80)	(12, 58)	(47, 6)	Tidak Valid
29	dokumen 29.pdf	0cbe89715456b85090f46d6fac779824bda8696dd20f4c30700fea43e9898bd4	(4, 16)	(19, 20)	(7, 2)	Tidak Valid
30	dokumen 30.pdf	48669eccede3b307b21d94e4391bd37c6e1647de3fcc26040549db7cb4929b01	(30, 41)	(47, 6)	(28, 4)	Tidak Valid

4.3 Evaluasi Hasil Pengujian

Aplikasi tanda tangan digital berbasis *Python* (PyQt5) telah diuji pada 30 dokumen PDF dengan empat tahapan utama, yaitu tahap penentuan titik, penentuan kunci, penandatanganan, dan verifikasi. Secara keseluruhan, aplikasi ini berfungsi dengan baik dalam setiap tahap, menunjukkan kinerja yang akurat dan efisien dalam proses tanda tangan digital menggunakan fungsi *hash* SHA-256 dan algoritma ECDSA.

Pada tahap penentuan titik, aplikasi berhasil menentukan titik-titik yang valid pada kurva eliptik sesuai dengan persamaan yang digunakan. Hasil yang diperoleh sesuai dengan teori matematika yang mendasari kurva eliptik dalam bidang hingga (*finite field*). Hal ini memastikan bahwa proses pembuatan kunci publik dan tanda tangan digital dilakukan pada domain yang benar, sehingga keamanan algoritma tetap terjaga.

Pada tahap penentuan kunci, aplikasi mampu menghasilkan pasangan kunci privat dan kunci publik dengan benar. Kunci publik yang digunakan harus konsisten untuk memastikan bahwa tanda tangan digital dapat diverifikasi dengan benar.

Pada tahap penandatanganan, aplikasi menghasilkan tanda tangan digital dalam bentuk pasangan nilai (r, s) yang sesuai dengan algoritma ECDSA. Selain itu, aplikasi juga menyimpan parameter yang digunakan dalam proses tanda tangan, kunci publik, dan tanda tangan digital ke dalam metadata dokumen. Parameter yang digunakan dalam proses tanda tangan dienkripsi agar tidak dapat diakses langsung oleh penerima tanda tangan, sehingga meningkatkan aspek keamanan dalam sistem.

Pada tahap verifikasi, aplikasi berhasil membaca metadata dokumen dan secara otomatis melakukan dekripsi terhadap parameter yang sebelumnya

dienkripsi. Parameter yang berhasil didekripsi tetap tersembunyi dalam jendela aplikasi untuk menjaga keamanan data. Sistem juga mampu mengisi kolom input kunci publik dan tanda tangan digital secara otomatis berdasarkan metadata dokumen yang diunggah. Proses verifikasi berjalan dengan baik, di mana aplikasi dapat menghitung nilai (r, s) dan membandingkannya dengan nilai verifikasi (v) untuk menentukan validitas tanda tangan. Hasil pengujian menunjukkan bahwa aplikasi konsisten memberikan *output* yang akurat dalam menentukan apakah tanda tangan digital valid atau tidak valid.

Dari hasil pengujian terhadap 30 dokumen, sebanyak 25 dokumen yang tidak mengalami perubahan dapat diverifikasi dengan tanda tangan digital yang valid. Sementara itu, lima dokumen lainnya mengalami perubahan pada berbagai elemen, seperti tanda tangan digital, kunci publik, atau isi dokumen. Perubahan ini bertujuan untuk menguji ketahanan sistem dalam mendeteksi integritas dokumen yang telah dimanipulasi. Jenis perubahan yang diterapkan pada dokumen-dokumen tersebut disajikan dalam Tabel 4.1.

Hasil pengujian menunjukkan bahwa aplikasi dapat secara akurat mendeteksi perubahan-perubahan tersebut. Jika kunci publik diubah, aplikasi tidak akan memproses perhitungan sehingga tidak menghasilkan nilai (r, s), menandakan bahwa tanda tangan digital tidak valid.

Dari segi kinerja, aplikasi memiliki proses yang relatif cepat dalam menghasilkan dan memverifikasi tanda tangan digital. Namun, terdapat tantangan dalam menghasilkan tanda tangan digital untuk setiap dokumen, karena nilai tanda tangan yang dihasilkan harus merupakan titik yang valid pada kurva eliptik dengan persamaan yang digunakan. Hal ini menyebabkan beberapa kendala teknis dalam

pembuatan tanda tangan yang valid untuk dokumen tertentu.

Dalam pengujian ini, dokumen yang digunakan hanya berbasis teks, tidak menggunakan elemen multimedia seperti gambar, tabel, atau grafik. Hal ini bertujuan untuk memastikan hasil *hash* yang stabil serta menghindari perubahan metadata otomatis yang dapat mempengaruhi integritas dokumen.

Secara keseluruhan, aplikasi ini berhasil memenuhi tujuan penelitian dengan akurasi tinggi dalam setiap tahapannya serta mampu mendeteksi perubahan yang berpengaruh terhadap keabsahan tanda tangan digital. Dengan implementasi yang kuat dan mekanisme keamanan yang diterapkan, aplikasi ini dapat menjadi alat yang andal dalam verifikasi tanda tangan digital berbasis kurva eliptik.

Dengan demikian, aplikasi ini telah berhasil mengimplementasikan tanda tangan digital menggunakan SHA-256 dan ECDSA dengan akurasi tinggi serta mekanisme verifikasi yang kuat. Hasil pengujian menunjukkan bahwa sistem mampu mendeteksi perubahan pada dokumen secara efektif, memastikan integritas dan keabsahan tanda tangan digital. Ke depan, pengembangan lebih lanjut dapat difokuskan pada peningkatan kompatibilitas dengan dokumen yang mengandung elemen multimedia serta optimalisasi kinerja aplikasi dalam pemrosesan tanda tangan digital.

4.4 Wujud Amanah pada Pembuatan Tanda Tangan Digital

Dalam konteks amanah terhadap sesama manusia, menjaga kepercayaan adalah hal yang sangat mendasar. Sebagaimana dijelaskan dalam Surah Al-Anfal ayat 27, Allah SWT berfirman:

يَأَيُّهَا الَّذِينَ آمَنُوا لَا تَخُونُوا اللَّهَ وَالرَّسُولَ وَتَخُونُوا أَمَانَاتِكُمْ وَأَنْتُمْ تَعْلَمُونَ

Artinya: ”Wahai orang-orang yang beriman, janganlah kamu mengkhianati

Allah SWT dan Rasul dan (juga) janganlah kamu mengkhianati amanat yang dipercayakan kepadamu, sedangkan kamu mengetahui" (QS. Al-Anfal: 27).

Ayat tersebut menekankan pentingnya kejujuran dan tanggung jawab dalam menjalankan amanah, termasuk saat menyampaikan informasi atau mandat kepada pihak lain.

Mandat terhadap sesama manusia, sebagaimana dibahas dalam subbab 2.8, merupakan salah satu dari tiga kategori amanah yang harus dipenuhi tanpa mengurangi hak-hak penerima yang sah. Dalam era digital, tanda tangan digital pada dokumen elektronik adalah salah satu cara untuk menjaga kepercayaan dan memenuhi kewajiban tersebut. Pengirim, sebagai pihak yang membawa amanah, membuat tanda tangan digital yang kemudian di verifikasi oleh penerima sebagai bukti autentikasi dan keabsahan dokumen. Dengan demikian, tanda tangan digital membantu memastikan bahwa isi dokumen sampai kepada penerima sesuai aslinya tanpa modifikasi.

Selain itu, penerapan tanda tangan digital juga mencerminkan komitmen untuk menjaga keabsahan dokumen, yang selaras dengan nilai-nilai amanah seperti memenuhi janji, menghormati hak milik orang lain, dan menyampaikan informasi yang benar. Ketika tanda tangan digital digunakan, pengirim menunjukkan bahwa ia memenuhi tanggung jawabnya kepada penerima dengan mengirimkan dokumen yang autentik. Menjaga kepercayaan dalam hal ini juga berarti tidak melakukan perubahan pada isi dokumen yang telah ditandatangani sehingga penerima dapat memvalidasi kebenarannya dengan tepat.

Penggunaan tanda tangan digital tidak hanya memastikan keabsahan informasi, tetapi juga mencerminkan prinsip transparansi dan akuntabilitas yang sangat dijunjung tinggi dalam Islam. Dalam pengiriman dokumen digital, setiap

tahap autentikasi dan verifikasi memberikan kejelasan mengenai asal dokumen dan menjaga ketelusuran informasi. Hal ini sejalan dengan ajaran Islam yang mengutamakan kejujuran dan kejelasan, sehingga dapat mengurangi risiko penipuan dan menjaga hak-hak penerima dengan penuh tanggung jawab.

Selanjutnya, tanda tangan digital juga menghormati hak penerima atas informasi yang valid dan terlindungi. Islam mengajarkan bahwa amanah berarti memberikan sesuatu kepada yang berhak dengan cara yang benar. Dengan menggunakan tanda tangan digital, pengirim dan penerima dapat mempercayai isi dokumen dengan keyakinan penuh bahwa informasi tersebut asli dan tidak mengalami perubahan. Penggunaan tanda tangan digital bukan hanya merupakan langkah teknis, melainkan wujud nyata dari pemenuhan amanah dalam Islam, yang mengedepankan penghormatan terhadap hak-hak setiap pihak dan menjaga kepercayaan secara utuh.

Penggunaan tanda tangan digital dalam dokumen elektronik mencerminkan kepatuhan terhadap prinsip menjaga privasi dan keamanan, yang merupakan bagian dari amanah dalam Islam. Islam menekankan pentingnya melindungi informasi yang diamanahkan kepada seseorang. Tanda tangan digital menjaga informasi agar aman dari akses atau perubahan yang tidak sah, serta memastikan hanya pihak berwenang yang dapat mengakses dan memverifikasi dokumen. Perlindungan ini merupakan bentuk nyata penerapan amanah, sehingga teknologi tanda tangan digital tidak hanya relevan secara hukum dan teknis, tetapi juga selaras dengan etika keagamaan yang menekankan tanggung jawab dalam menjaga kepercayaan dan hak-hak orang lain.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan rumusan masalah dan pembahasan yang telah disajikan, kesimpulan yang dapat diambil adalah sebagai berikut:

1. Implementasi tanda tangan digital menggunakan fungsi *hash* SHA-256 dan algoritma ECDSA telah berhasil dilakukan melalui empat tahapan utama, yaitu penentuan titik pada kurva eliptik, pembangkitan pasangan kunci, proses penandatanganan, dan verifikasi tanda tangan. Hasil implementasi menunjukkan bahwa setiap tahap berjalan sesuai dengan prinsip matematika dalam kriptografi kurva eliptik. Penentuan titik memastikan bahwa himpunan titik yang digunakan valid dalam bidang hingga, sedangkan pembangkitan kunci menghasilkan pasangan kunci privat dan publik yang memenuhi hubungan matematis dalam ECDSA. Pada tahap penandatanganan, tanda tangan digital dalam bentuk pasangan nilai (r, s) dihasilkan berdasarkan fungsi *hash* SHA-256 dan operasi aritmatika kurva eliptik, sehingga memastikan tanda tangan unik untuk setiap pesan.
2. Hasil verifikasi tanda tangan digital menunjukkan bahwa tanda tangan yang dihasilkan dapat diuji keabsahannya dengan membandingkan nilai (r, s) dengan hasil perhitungan verifikasi. Pengujian membuktikan bahwa sistem mampu memverifikasi tanda tangan pada dokumen yang valid serta mendeteksi perubahan pada dokumen, kunci publik, atau tanda tangan digital, sehingga memastikan integritas pesan. Jika terjadi perubahan pada

elemen-elemen tersebut, sistem secara otomatis menolak tanda tangan digital sebagai tidak valid. Dengan demikian, implementasi ini telah memenuhi prinsip keamanan digital, yaitu auntentikasi, integritas, dan non-repudiasi. Dibandingkan dengan penelitian sebelumnya, penelitian ini memiliki keunggulan dalam penggunaan algoritma SHA-256 yang lebih aman dibandingkan SHA-1 atau MD5, serta pendekatan matematis yang lebih mendalam dalam implementasi ECDSA. Selain itu, pengujian dilakukan pada jumlah dokumen yang lebih banyak untuk menguji keandalannya. Penelitian ini juga mengintegrasikan prinsip keamanan digital dengan konsep amanah dalam islam, yang menekankan pentingnya menjaga keabsahan, kepercayaan dan integritas dalam sistem tanda tangan digital. Dengan demikian, penelitian ini membuktikan bahwa tanda tangan digital berbasis kurva eliptik dapat diterapkan dengan tingkat keamanan yang tinggi dan kinerja yang baik, serta mencapai tujuan utama penelitian dalam implementasi dan verifikasinya.

5.2 Saran

1. Disarankan untuk mengembangkan dukungan algoritma tanda tangan digital yang lebih kuat serta mengoptimalkan perhitungan titik valid pada kurva eliptik guna meningkatkan keamanan dan efisiensi sistem.
2. Perlu dilakukan integrasi dengan sistem manajemen dokumen atau platform berbasis web agar sistem lebih fleksibel dan mudah digunakan.
3. Disarankan menerapkan enkripsi tambahan pada metadata untuk meningkatkan perlindungan dari manipulasi tanda tangan digital.

DAFTAR PUSTAKA

Alur Pembentukan Word untuk Prepare Message Schedule [Gambar]. (n.d.). Diakses pada 3 November 2024, dari https://images.app.goo.gl/bq7AbqVZACsxyh2W9

Arwa, N., Aminuddin, & Arifianto, S. (2021). *Implementasi tanda tangan digital menggunakan ECDSA (Studi kasus: Jurnal tipe file PDF)*. Universitas Muhammadiyah Malang.

Aziz, A. A. (2009). *Implementasi tanda tangan digital menggunakan metode Ong-Schnorr-Shamir dan Euclidean pada teks*. Universitas Indonesia.

Fachruddin Hs. (n.d.). *Ensiklopedia pengetahuan Al-Qur'an dan Hadits* (hlm. 15).

Gallian, J. A. (2021). *Contemporary abstract algebra* (10th ed.). CRC Press.

Hamidy, Z. (1981). Terjemah Hadist Shahih Bukhari. Jakarta: PT Bumirestu.

Hamka. (1991). *Tafsir Al-Azhar*. Jakarta: Pustaka.

Hardiningsih, A. H. R. (2021). *Implementasi fungsi hash MD5 dan kriptografi algoritma RSA pada pembuatan tanda tangan digital*. Universitas Muhammadiyah Purwokerto.

Insani, A. Y. (2008). *Proteksi akses file executable menggunakan sistem keamanan teknologi USB Flash Disk* [Undergraduate thesis, Universitas Komputer Indonesia].

Judson, T. W. (2020). *Abstract algebra: Theory and applications*. Stephen F. Austin State University.

Kementerian Agama Republik Indonesia. (2019). *Al-Qur'an dan terjemahannya* [PDF]. Retrieved from https://archive.org/details/2019-al-quran-dan-terjemahnya-edisi-2019 (diakses 23 September 2024).

Kementerian Pendidikan dan Kebudayaan. (2016). *Kamus besar bahasa Indonesia*.

https://kbbi.kemdikbud.go.id/entri/amanah (diakses 24 September 2024).

Mallouli, F., Hellal, A., Saeed, N. S., & Alzahrani, F. A. (2019). A survey on cryptography: Comparative study between RSA vs ECC algorithms, and RSA vs El-Gamal algorithms. *In Proceedings of the 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud 2019) and 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom 2019)* (pp. 173–176). https://doi.org/10.1109/CSCloud/EdgeCom.2019.00022.

Mankar, R. V., & Nipanikar, S. I. (2013). *C implementation of SHA-256 algorithm*. Pune University.

Menezes, A., Van Oorschot, P., & Vanstone, S. (1996). *Handbook of applied cryptography*. CRC Press.

Munawwir, A. W. (1997). *Kamus Al-Munawwir Arab-Indonesia terlengkap*. Surabaya: Pustaka Progressif.

Munir, R. (2019). *Kriptografi*. Bandung: Informatika.

Proses Fungsi Kompresi [Gambar]. (n.d.). Diakses pada 3 November 2024, dari https://images.app.goo.gl/1F4Rd3iG4hCZTkmm6

Q Shaleh, et al. (2009). *Asbabul Nuzul; Latar belakang historis turunnya ayat-ayat al-Qur'an*. Bandung: Penerbit Diponegoro.

Ramadha, A. (2018). *Implementasi tanda tangan digital dengan menggunakan elliptic curve untuk validasi one-time password*. Informatika, Institut Teknologi Bandung.

Rodriguez-Henriquez, F., et al. (2006). *Cryptographic algorithms on reconfigurable hardware*. New York: Springer.

Schneier, B. (1996). *Applied cryptography, second edition: Protocols, algorithms, and source code in C*. John Wiley & Sons, Inc.

Shaleh, M. Q., dkk. (2009). Tafsir Al-Misbah. Jakarta: Lentera Hati.

Skema Digital Signature [Gambar]. (n.d.). Diakses pada 3 November 2024, dari https://images.app.goo.gl/K34yEG4WcqaMyRJW7

Triwinarko, A. (2005). *Elliptic Curve Digital Signature Algorithm (ECDSA)*.

LAMPIRAN

LAMPIRAN 1. Tabel ASCII

Dec	Hex	Bin	Char	Dec	Hex	Bin	Char
0	0x00	00000000	(NUL)	64	0x40	01000000	@
1	0x01	00000001	(SOH)	65	0x41	01000001	A
2	0x02	00000010	(STX)	66	0x42	01000010	B
3	0x03	00000011	(ETX)	67	0x43	01000011	C
4	0x04	00000100	(EOT)	68	0x44	01000100	D
5	0x05	00000101	(ENQ)	69	0x45	01000101	E
6	0x06	00000110	(ACK)	70	0x46	01000110	F
7	0x07	00000111	(BEL)	71	0x47	01000111	G
8	0x08	00001000	(BS)	72	0x48	01001000	H
9	0x09	00001001	(TAB)	73	0x49	01001001	I
10	0x0A	00001010	(LF)	74	0x4A	01001010	J
11	0x0B	00001011	(VT)	75	0x4B	01001011	K
12	0x0C	00001100	(FF)	76	0x4C	01001100	L
13	0x0D	00001101	(CR)	77	0x4D	01001101	M
14	0x0E	00001110	(SO)	78	0x4E	01001110	N
15	0x0F	00001111	(SI)	79	0x4F	01001111	O
16	0x10	00010000	(DLE)	80	0x50	01010000	P
17	0x11	00010001	(DC1)	81	0x51	01010001	Q
18	0x12	00010010	(DC2)	82	0x52	01010010	R
19	0x13	00010011	(DC3)	83	0x53	01010011	S
20	0x14	00010100	(DC4)	84	0x54	01010100	T
21	0x15	00010101	(NAK)	85	0x55	01010101	U
22	0x16	00010110	(SYN)	86	0x56	01010110	V
23	0x17	00010111	(ETB)	87	0x57	01010111	W
24	0x18	00011000	(CAN)	88	0x58	01011000	X
25	0x19	00011001	(EM)	89	0x59	01011001	Y
26	0x1A	00011010	(SUB)	90	0x5A	01011010	Z
27	0x1B	00011011	(ESC)	91	0x5B	01011011	[
28	0x1C	00011100	(FS)	92	0x5C	01011100	\
29	0x1D	00011101	(GS)	93	0x5D	01011101	]
30	0x1E	00011110	(RS)	94	0x5E	01011110	^
31	0x1F	00011111	(US)	95	0x5F	01011111	_
32	0x20	00100000	space	96	0x60	01100000	`
33	0x21	00100001	!	97	0x61	01100001	a
34	0x22	00100010	"	98	0x62	01100010	b

Dec	Hex	Bin	Char	Dec	Hex	Bin	Char
35	0x23	00100011	#	99	0x63	01100011	c
36	0x24	00100100	$	100	0x64	01100100	d
37	0x25	00100101	%	101	0x65	01100101	e
38	0x26	00100110	&	102	0x66	01100110	f
39	0x27	00100111	'	103	0x67	01100111	g
40	0x28	00101000	(	104	0x68	01101000	h
41	0x29	00101001	)	105	0x69	01101001	i
42	0x2A	00101010	*	106	0x6A	01101010	j
43	0x2B	00101011	+	107	0x6B	01101011	k
44	0x2C	00101100	,	108	0x6C	01101100	l
45	0x2D	00101101	-	109	0x6D	01101101	m
46	0x2E	00101110	.	110	0x6E	01101110	n
47	0x2F	00101111	/	111	0x6F	01101111	o
48	0x30	00110000	0	112	0x70	01110000	p
49	0x31	00110001	1	113	0x71	01110001	q
50	0x32	00110010	2	114	0x72	01110010	r
51	0x33	00110011	3	115	0x73	01110011	s
52	0x34	00110100	4	116	0x74	01110100	t
53	0x35	00110101	5	117	0x75	01110101	u
54	0x36	00110110	6	118	0x76	01110110	v
55	0x37	00110111	7	119	0x77	01110111	w
56	0x38	00111000	8	120	0x78	01111000	x
57	0x39	00111001	9	121	0x79	01111001	y
58	0x3A	00111010	:	122	0x7A	01111010	z
59	0x3B	00111011	;	123	0x7B	01111011	{
60	0x3C	00111100	<	124	0x7C	01111100	|
61	0x3D	00111101	0	125	0x7D	01111101	}
62	0x3E	00111110	>	126	0x7E	01111110	~
63	0x3F	00111111	?	127	0x7F	01111111	(DEL)

LAMPIRAN 2. *Script User Interface* Tanda Tangan Digital

```
import random
import json
import sys
import os
import math
from PyQt5 import QtWidgets, QtCore, QtGui
from PyQt5.QtWidgets import (QApplication, QMainWindow, QVBoxLayout,\
                             QHBoxLayout,QLabel, QLineEdit, QPushButton,\
                            QWidget, QTabWidget,QTextEdit, QTextBrowser, \
                            QFileDialog, QFormLayout, QGroupBox)
from PyQt5.QtGui import QFont
from PyQt5.QtCore import Qt
from cryptography.hazmat.primitives.asymmetric import ec
from ecdsa import SigningKey, VerifyingKey, NIST384p
from ecdsa import VerifyingKey, BadSignatureError
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
import hashlib
from PyPDF2 import PdfReader
from math import gcd
from PyPDF2 import PdfWriter
from datetime import datetime
import pikepdf
from pikepdf import Pdf, Page
import re
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
import base64
# Window untuk penentuan titik-titik
class CurvePointsWindow(QWidget):
    def __init__(self, main_window):
        super().__init__()
        self.main_window = main_window  # Referensi ke MainWindow

        # Atur ukuran tetap yang lebih besar agar tulisan tidak terpotong
        self.setFixedSize(700, 800)

        # Layout utama vertikal
        layout = QVBoxLayout()
        layout.setContentsMargins(20, 20, 20, 20)  # Margin sekitar konten

        # Judul aplikasi dengan font lebih besar
        self.title_label = QLabel("Penentuan Titik Kurva Eliptik")
        self.title_label.setFont(QFont('Arial', 18, QFont.Bold))
        self.title_label.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.title_label)
```

```
        # Input untuk a, b, dan p dengan label yang seragam
        self.a_input = self.create_labeled_input("Masukkan nilai a:", layout)
        self.b_input = self.create_labeled_input("Masukkan nilai b:", layout)
        self.p_input = self.create_labeled_input("Masukkan modulus (p):", layout)

        # Tombol Hitung
        self.calculate_button = QPushButton("Hitung Titik-Titik")
        self.calculate_button.setFont(QFont('Arial', 14))
        self.calculate_button.setFixedSize(200, 50)
        self.calculate_button.setStyleSheet("background-color: #4CAF50; color:\
                                             white; border-radius: 8px;")
        self.calculate_button.clicked.connect(self.calculate_points)

        # Label Persamaan Kurva
        self.equation_label = QLabel("Persamaan Kurva: y² ≡ x³ + ax + b mod p")
        self.equation_label.setFont(QFont('Arial', 12, QFont.StyleItalic))
        self.equation_label.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.equation_label)

        # Area hasil dan jumlah titik
        self.result_area = QTextEdit()
        self.result_area.setFont(QFont('Arial', 12))
        self.result_area.setReadOnly(True)
        self.result_area.setFixedHeight(150)
        self.result_area.setPlaceholderText("Hasil titik-titik akan muncul di sini.")

        self.point_count_label = QLabel("Jumlah titik (n): -")
        self.point_count_label.setFont(QFont('Arial', 14))
        self.point_count_label.setAlignment(Qt.AlignCenter)

        # Menyusun semua komponen ke dalam layout utama
        layout.addWidget(self.result_area)
        layout.addWidget(self.point_count_label)
        layout.addWidget(self.calculate_button, alignment=Qt.AlignCenter)

    # Atur layout utama
    self.setLayout(layout)

def create_labeled_input(self, label_text, layout):
    """Fungsi untuk membuat input field dengan label."""
    container = QWidget()
    h_layout = QHBoxLayout()
    h_layout.setContentsMargins(0, 5, 0, 5)

    label = QLabel(label_text)
    label.setFont(QFont('Arial', 14))
    input_field = QLineEdit()
    input_field.setFont(QFont('Arial', 12))
    input_field.setFixedWidth(400)

    input_field.setStyleSheet("""
        QLineEdit {
            border: 1px solid #A9A9A9;
            border-radius: 5px;
            padding: 5px;
            background: qlineargradient(spread:pad, x1:0, y1:0, x2:1,
                                        stop:0 #FFFFFF, stop:1 #F0F0F0);
        }
```

```
        QLineEdit:focus {
            border: 1px solid #4CAF50;
            background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,\
                stop:0 #FFFFFF, stop:1 #E0F7E0);
        }
    """)

    h_layout.addWidget(label)
    h_layout.addWidget(input_field, alignment=Qt.AlignRight)
    container.setLayout(h_layout)

    # Tambahkan ke layout utama
    layout.addWidget(container)
    return input_field

def calculate_points(self):
    # Ambil nilai a, b, dan modulus (p)
    a_text = self.a_input.text().strip()
    b_text = self.b_input.text().strip()
    p_text = self.p_input.text().strip()

    # Validasi input
    if not (a_text.isdigit() and b_text.isdigit() and p_text.isdigit()):
        self.result_area.setText("Nilai a, b, dan p harus berupa angka.")
        return

    a = int(a_text)
    b = int(b_text)
    p = int(p_text)

    # Tampilkan persamaan kurva dengan nilai yang dimasukkan
    self.equation_label.setText(f"Persamaan Kurva: y² ≡ x³ + {a}x + {b} mod {p}")


    # Hitung titik-titik (x, y) di GF(p)
    self.points = []  # Reset daftar titik
    for x in range(p):
        rhs = (x**3 + a * x + b) % p
        for y in range(p):
            if (y**2 % p) == rhs:
                self.points.append((x, y))

    # Format hasil dan tampilkan
    if self.points:
        points_str = ", ".join([f"({x},{y})" for x, y in self.points])
        self.result_area.setText(f"Titik-titik: {points_str}")
        self.point_count_label.setText(f"Jumlah titik (n): {len(self.points)}")

        # Simpan persamaan dan titik-titik untuk diakses di tab lain
        self.main_window.set_equation(f"y² ≡ x³ + {a}x + {b} mod {p}", self.points)
    else:
        self.result_area.setText("Tidak ada titik yang ditemukan.")
        self.point_count_label.setText("Jumlah titik (n): 0")
```

```
    def calculate_curve_order(self, a, b, p):
        # Menghitung order kurva eliptik
        order = 0
        for x in range(p):
            rhs = (x**3 + a * x + b) % p
            for y in range(p):
                if (y**2 % p) == rhs:
                    order += 1
        return order

    def get_curve_order(self):
        # Mendapatkan parameter kurva untuk menghitung order
        a_text = self.a_input.text().strip()
        b_text = self.b_input.text().strip()
        p_text = self.p_input.text().strip()

        if a_text.isdigit() and b_text.isdigit() and p_text.isdigit():
            a = int(a_text)
            b = int(b_text)
            p = int(p_text)
            return self.calculate_curve_order(a, b, p)
        else:
            return None  # Atau bisa juga raise exception jika diinginkan


# Window untuk penentuan key
class KeyGenerationWindow(QWidget):
    def __init__(self, main_window, curve_points_window):
        super().__init__()
        self.main_window = main_window  # Referensi ke MainWindow
        self.curve_points_window = curve_points_window  # ReferensikeCurvePointsWindow

        layout = QVBoxLayout()

        self.title_label = QLabel("Generate Keys (Kunci Publik & Privat)")
        self.title_label.setFont(QFont('Arial', 17, QFont.Bold))
        self.title_label.setAlignment(Qt.AlignCenter)

        # Label dan input untuk kunci privat (dA)
        self.da_label = QLabel("Masukkan dA (kunci privat, 1 ≤ dA < n):")
        self.da_label.setStyleSheet("font-size: 16px; font-weight: bold; color: #333;")
        self.da_input = QLineEdit()
        self.da_input.setStyleSheet("""
            QLineEdit {
                border: 1px solid #A9A9A9;
                border-radius: 5px;
                padding: 5px;
                background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,\
                                stop:0 #FFFFFF, stop:1 #F0F0F0);
                font-size: 14px;
            }
            QLineEdit:focus {
                border: 1px solid #4CAF50;
                background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,\
                                stop:0 #FFFFFF, stop:1 #E0F7E0);
            }
        """)
```

```
# Label dan input untuk titik G (Generator point)
self.g_label = QLabel("Pilih Titik G (Generator):")
self.g_label.setStyleSheet("font-size: 16px; font-weight: bold; color: #333;")
self.gx_input = QLineEdit()
self.gx_input.setStyleSheet("""
    QLineEdit {
        border: 1px solid #A9A9A9;
        border-radius: 5px;
        padding: 8px;
        background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,\
                    stop:0 #FFFFFF, stop:1 #F0F0F0);
        font-size: 14px;
    }
    QLineEdit:focus {
        border: 1px solid #4CAF50;
        background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,\
                    stop:0 #FFFFFF, stop:1 #E0F7E0);
    }
""")
self.gy_input = QLineEdit()
self.gy_input.setStyleSheet("""
    QLineEdit {
        border: 1px solid #A9A9A9;
        border-radius: 5px;
        padding: 8px;
        background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,\
                    stop:0 #FFFFFF, stop:1 #F0F0F0);
        font-size: 14px;
    }
    QLineEdit:focus {
        border: 1px solid #4CAF50;
        background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,\
                    stop:0 #FFFFFF, stop:1 #E0F7E0);
    }
""")

# Tombol untuk menghitung kunci publik
self.calculate_button = QPushButton("Hitung QA (Public Key)")
self.calculate_button.setFont(QFont('Arial', 12, QFont.Bold))
self.calculate_button.setFixedSize(200, 50)
self.calculate_button.setStyleSheet("""
    background-color: #4CAF50;
    color: white;
    border-radius: 8px;
    border: none;
    font-weight: bold;
    font-size: 15px;
    padding: 10px;
    text-align: center;
""")
self.calculate_button.clicked.connect(self.generate_public_key)
```

```
    # Area untuk menampilkan hasil kunci publik
    self.result_area = QTextEdit("QA (Public Key) akan muncul di sini.")
    self.result_area.setStyleSheet("""
        background-color: #f1f1f1;
        border: 1px solid #ccc;
        border-radius: 5px;
        height: 100px;
        font-size: 10pt;
    """)

    # Layout untuk form input
    form_layout = QVBoxLayout()
    form_layout.addWidget(self.title_label)
    form_layout.addSpacing(20)
    form_layout.addWidget(self.da_label)
    form_layout.addWidget(self.da_input)
    form_layout.addSpacing(10)
    form_layout.addWidget(self.g_label)
    form_layout.addWidget(QLabel("Gx:"))
    form_layout.addWidget(self.gx_input)
    form_layout.addWidget(QLabel("Gy:"))
    form_layout.addWidget(self.gy_input)
    form_layout.addWidget(self.calculate_button)

    # Menambahkan form_layout (input dan label) ke layout utama
    layout.addLayout(form_layout)
    layout.addWidget(self.result_area)

    # Set layout utama
    self.setLayout(layout)
```

```
def generate_public_key(self):
    try:
        da = int(self.da_input.text())  # Kunci privat dA
        gx = int(self.gx_input.text())  # Gx
        gy = int(self.gy_input.text())  # Gy

        # Ambil parameter kurva dari MainWindow
        a, b, p = self.main_window.get_curve_params()
        n = self.curve_points_window.get_curve_order()

        # Cek apakah dA valid dan dalam rentang yang benar
        if not (1 <= da < n):
            self.result_area.setText(f"Kunci privat dA harus berada\
                                     di antara 1 dan n-1 (1 ≤ dA < {n}).")
            return

        # Simpan titik G yang dipilih dalam file JSON
        G = (gx, gy)

        # Perhitungan QA = dA * G
        qa = self.point_multiply(da, G, a, b, p)
```

```
            # Menampilkan hasil
            if self.is_point_on_curve(qa[0], qa[1], a, b, p):
                public_key = f"QA = {qa}\nTitik kunci publik berada di dalam kurva."
            else:
                public_key = f"QA = {qa}\nTitik kunci publik TIDAK berada di dalam kurva."

            self.result_area.setText(public_key)

        except ValueError:
            self.result_area.setText("Masukkan angka yang valid untuk\
                                     kunci privat dan titik G (Gx, Gy).")
        except Exception as e:
            self.result_area.setText(f"Terjadi kesalahan: {str(e)}")

    def is_point_on_curve(self, x, y, a, b, p):
        # Cek apakah titik (x, y) memenuhi persamaan kurva eliptik
        return (y * y) % p == (x * x * x + a * x + b) % p

    def point_multiply(self, k, point, a, b, p):
        qa = (0, 0)  # Titik tak terdefinisi
        temp_point = point

        while k > 0:
            if k % 2 == 1:
                qa = self.point_add(qa, temp_point, a, b, p)
            temp_point = self.point_double(temp_point, a, b, p)
            k //= 2

        return qa


    def point_add(self, p1, p2, a, b, p):
        if p1 == (0, 0):
            return p2
        if p2 == (0, 0):
            return p1

        x1, y1 = p1
        x2, y2 = p2

        if x1 == x2 and y1 == y2:
            return self.point_double(p1, a, b, p)

        if x1 == x2:
            return (0, 0)

        lam = (y2 - y1) * pow((x2 - x1), p - 2, p) % p

        x3 = (lam**2 - x1 - x2) % p
        y3 = (lam * (x1 - x3) - y1) % p

        return (x3, y3)
```

```
    def point_double(self, point, a, b, p):
        x, y = point

        if y == 0:
            return (0, 0)

        lam = (3 * x**2 + a) * pow((2 * y), p - 2, p) % p

        x3 = (lam**2 - 2 * x) % p
        y3 = (lam * (x - x3) - y) % p

        return (x3, y3)


# Window untuk penandatanganan (Signing)
class SigningWindow(QtWidgets.QWidget):
    def __init__(self, main_window):
        super().__init__()
        self.main_window = main_window
        self.pdf_path = None
        self.hashed_message = None
        self.encryption_key = get_random_bytes(16)  # Kunci 128-bit untuk AES
        self.cipher = AES.new(self.encryption_key, AES.MODE_CBC)
        layout = QVBoxLayout()

        # Title
        self.title_label = QLabel("Proses Penandatanganan Digital (ECDSA)")
        self.title_label.setFont(QFont('Arial', 17, QFont.Bold))
        self.title_label.setAlignment(Qt.AlignCenter)
        self.title_label.setStyleSheet("margin-bottom: 20px;")
        layout.addWidget(self.title_label)

        # File PDF Selection
        self.pdf_label = QLabel("Pilih dokumen PDF yang ingin di-hash:")
        self.pdf_label.setStyleSheet("font-size: 17px; font-weight: bold;\
                                     color: #333;")
        self.file_button = self.create_button("Pilih File PDF", "#FF9800")
        self.file_button.clicked.connect(self.select_pdf_file)
        self.file_path_display = QLabel("Belum ada file yang dipilih.")
        self.file_path_display.setStyleSheet("font-size: 15px; color: #555;")
        layout.addWidget(self.pdf_label)
        layout.addWidget(self.file_button)
        layout.addWidget(self.file_path_display)

        # Curve Parameters
        self.add_curve_inputs(layout)

        # Private Key Input
        self.da_label = QLabel("Masukkan dA (private key):")
        self.da_label.setStyleSheet("font-size: 16px; font-weight: bold;\
                                    color: #333;")
        self.da_input = self.create_input_field()
        layout.addWidget(self.da_label)
        layout.addWidget(self.da_input)
```

```
        # Input untuk Public Key
        self.public_key_label = QLabel("Masukkan Public Key (Dalam format x,y):")
        self.public_key_label.setStyleSheet("font-size: 16px; font-weight: bold;\
                                            color: #333;")
        self.public_key_input = QLineEdit()
        self.public_key_input = self.create_input_field()
        layout.addWidget(self.public_key_label)
        layout.addWidget(self.public_key_input)

        # Nonce k
        self.k_label = QLabel("Masukkan k (nonce acak):")
        self.k_label.setStyleSheet("font-size: 16px; font-weight: bold; color: #333;")
        self.k_input = self.create_input_field()
        layout.addWidget(self.k_label)
        layout.addWidget(self.k_input)

        # Generator Point G
        self.G_label = QLabel("Masukkan G (Generator point dalam format x,y):")
        self.G_label.setStyleSheet("font-size: 16px; font-weight: bold; color: #333;")
        self.G_input = self.create_input_field()
        layout.addWidget(self.G_label)
        layout.addWidget(self.G_input)

        # Order n
        self.n_label = QLabel("Masukkan n (order kurva):")
        self.n_label.setStyleSheet("font-size: 16px; font-weight: bold; color: #333;")
        self.n_input = self.create_input_field()
        layout.addWidget(self.n_label)
        layout.addWidget(self.n_input)
        # Hashing Button and Output
        self.hash_button = self.create_button("Klik Pesan untuk Hashing", "#4CAF50")
        self.message_digest_area = self.create_text_browser()
        self.hash_button.clicked.connect(self.hash_message)
        layout.addWidget(self.hash_button)
        layout.addWidget(self.message_digest_area)

        # Sign Button and Signature Output
        self.sign_button = self.create_button("Tandatangani Pesan", "#2196F3")
        self.signature_area = self.create_text_browser()
        self.sign_button.clicked.connect(self.sign_message)
        layout.addWidget(self.sign_button)
        layout.addWidget(self.signature_area)

        # Download Signed PDF Button
        self.download_button = self.create_button\
            ("Download PDF yang Ditandatangani", "#9C27B0")
        self.download_button.setEnabled(False)
        self.download_button.clicked.connect(self.download_signed_pdf)
        layout.addWidget(self.download_button)

        # Set the layout
        self.setLayout(layout)

    def create_input_field(self):
        input_field = QLineEdit()
        input_field.setStyleSheet("""
            QLineEdit {
                border: 1px solid #A9A9A9;
                border-radius: 5px;
                padding: 5px;
                background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,\
                                stop:0 #FFFFFF, stop:1 #F0F0F0);
                font-size: 14px;
            }
```

```
        QLineEdit:focus {
            border: 1px solid #4CAF50;
            background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1, \
                        stop:0 #FFFFFF, stop:1 #E0F7E0);
        }
    """)
    return input_field

def create_button(self, text, color):
    button = QPushButton(text)
    button.setStyleSheet(f"""
        background-color: {color};
        color: white;
        font-size: 18px;
        padding: 5px;
        border-radius: 8px;
        border: none;
    """)
    return button

def create_text_browser(self):
    text_browser = QTextBrowser()
    text_browser.setStyleSheet("""
        background-color: #f1f1f1;
        border: 1px solid #ccc;
        border-radius: 5px;
        height: 100px;
        font-size: 15px;
    """)
    return text_browser

def add_curve_inputs(self, layout):
    curve_inputs_layout = QVBoxLayout()
    curve_label = QLabel("Masukkan parameter kurva:")
    curve_label.setStyleSheet("font-size: 16px; font-weight: bold;\
                              color: #333;")
    curve_inputs_layout.addWidget(curve_label)

    input_layout = QHBoxLayout()
    self.a_input = self.create_input_with_label("a:", input_layout)
    self.b_input = self.create_input_with_label("b:", input_layout)
    self.p_input = self.create_input_with_label\
                   ("p (modulus):", input_layout)

    curve_inputs_layout.addLayout(input_layout)
    layout.addLayout(curve_inputs_layout)
```

```
def create_input_with_label(self, label_text, layout):
    label = QLabel(label_text)
    input_field = self.create_input_field()
    layout.addWidget(label)
    layout.addWidget(input_field)
    return input_field


def select_pdf_file(self):
    """Memilih file PDF untuk di-hash."""
    options = QFileDialog.Options()
    file_path, _ = QFileDialog.getOpenFileName(
        self, "Pilih File PDF", "", "PDF Files (*.pdf);;All Files (*)",\
            options=options
    )
    if file_path:
        self.pdf_path = file_path
        self.file_path_display.setText(file_path)

def hash_message(self):
    """Meng-hash konten PDF menggunakan SHA-256."""
    try:
        if not self.pdf_path:
            self.message_digest_area.setText\
                ("Pilih file PDF terlebih dahulu.")
            return

        # Membaca file PDF
        with open(self.pdf_path, "rb") as pdf_file:
            pdf_reader = PdfReader(pdf_file)
            text_content = ""
            for page in pdf_reader.pages:
                text_content += page.extract_text() or ""

        # Meng-hash konten PDF
        self.hashed_message = hashlib.sha256(text_content.encode()).\
                                hexdigest()
        self.message_digest_area.setText\
            (f"Message Digest (SHA-256): {self.hashed_message}")

    except Exception as e:
        self.hashed_message = ""
        self.message_digest_area.setText\
            (f"Kesalahan dalam membaca atau memproses file PDF: {e}")

def sign_message(self):
    """Menandatangani pesan menggunakan ECDSA."""
    if not self.hashed_message:
        self.signature_area.setText\
            ("Hash pesan terlebih dahulu sebelum menandatangani.")
        return
```

```
message = self.hashed_message
print(f"Message digest yang digunakan untuk signing:\
    {message}")  # Debugging message digest
a_text = self.a_input.text()
b_text = self.b_input.text()
p_text = self.p_input.text()
da_text = self.da_input.text()
G_text = self.G_input.text()
n_text = self.n_input.text()
k_text = self.k_input.text()
public_key_text = self.public_key_input.text()

print(f"a_text: {a_text}, b_text: {b_text}, p_text: {p_text},\
    G_text: {G_text}, n_text: {n_text}, k_text: {k_text},\
        public_key_text: {public_key_text}")

if not a_text or not b_text or not p_text or not da_text \
    or not G_text or not n_text or not k_text\
    or not public_key_text:
    self.signature_area.setText\
    ("Masukkan semua input yang diperlukan: a, b, p, private key,\
     G, n, k, dan kunci publik.")
    return

try:
    a = int(a_text)
    b = int(b_text)
    p = int(p_text)
    da = int(da_text)
    G = tuple(map(int, G_text.split(',')))
    n = int(n_text)
    k = int(k_text) if k_text else self.generate_random_k(n)
    public_key = tuple(map(int, public_key_text.split(',')))

    print(f"G: {G}, public_key: {public_key}")

    if len(G) != 2 or len(public_key) != 2:
        raise ValueError("Titik G dan kunci publik harus \
                         dalam format (x,y).")

    self.G = G  # Simpan titik G di atribut

    # Verifikasi apakah G berada pada kurva eliptik
    if not self.is_point_on_curve(G[0], G[1], a, b, p):
        self.signature_area.setText("Titik G tidak valid pada kurva.")
        return

    # Cek GCD untuk k dan n
    if math.gcd(k, n) != 1:
        print("Nilai k tidak dapat di-inversi untuk modulus n.")
        self.signature_area.setText("Nilai k tidak valid, ulangi \
                                    dengan k yang berbeda.")
        return
    else:
        print("Nilai k dapat di-inversi.")
```

```
            # Menghitung r
            kG = self.point_multiply(k, G, a, b, p, n)
            r = kG[0] % n

            # Jika r == 0, ulangi penandatanganan
            if r == 0:
                self.signature_area.setText("Nilai r tidak valid, ulangi \
                                            dengan k yang berbeda.")
                return

            # Menghitung s
            k_inv = pow(k, -1, n)
            s = (k_inv * (int(self.hashed_message, 16) + da * r)) % n
            print(f"Nilai s yang dihitung: {s}")

            # Jika s == 0, ulangi penandatanganan
            if s == 0:
                self.signature_area.setText("Nilai s tidak valid, ulangi \
                                            dengan k yang berbeda.")
                return

            # Menyimpan metadata tanda tangan digital di atribut
            self.signature_metadata = {
                'a': a,
                'b': b,
                'p': p,
                'G': G,
                'n': n,
                'k': k,
                'r': r,
                's': s,
                'public_key': public_key
            }
        # Memeriksa apakah titik (r, s) berada di kurva eliptik
        if self.is_point_on_curve(r, s, a, b, p):
            self.save_signature_metadata(r, s, public_key, a, b, p, G, n, k)
            self.signature_area.setText(f"Tanda tangan digital: r = {r}, \
                                        s = {s} (Titik berada di kurva \
                                            eliptik)")
            self.download_button.setEnabled(True)
        else:
            self.signature_area.setText(f"Tanda tangan digital: r = {r},\
                                        s = {s} (Titik tidak berada \
                                            di kurva eliptik)")

    except ValueError as ve:
        self.signature_area.setText(f"Kesalahan input: {ve}")
    except Exception as e:
        self.signature_area.setText(f"Kesalahan saat menandatangani \
                                    pesan: {e}")


def encrypt_metadata(self, data: str, key: bytes) -> str:
    """Mengenkripsi data menggunakan AES (CBC mode)."""
    cipher = AES.new(key, AES.MODE_CBC)
    # Padding data untuk memenuhi ukuran blok AES
    padded_data = pad(data.encode(), AES.block_size)
    encrypted_data = cipher.encrypt(padded_data)
    # Menggabungkan IV dan data terenkripsi
    encrypted_data_with_iv = cipher.iv + encrypted_data
    # Encode hasilnya dalam base64 untuk penyimpanan yang mudah
    return base64.b64encode(encrypted_data_with_iv).decode()
```

```
def save_signature_metadata(self, r, s, public_key, a, b, p, G, n, k):
    """Menyimpan metadata tanda tangan ke dalam PDF dengan sebagian \
        terenkripsi menggunakan AES."""
    try:
        # Kunci AES (harusnya disimpan dengan aman, misalnya di \
        #environment variable atau keystore)
        encryption_key = get_random_bytes(32) # Kunci AES 256-bit
        # Encode ke base64
        encryption_key_encoded = base64.b64encode(encryption_key).decode()

        # Menyusun data yang akan disembunyikan (parameter kurva eliptik)
        sensitive_data = f'a={a};b={b};p={p};G={G};n={n};k={k}'

        # Mengenkripsi data sensitif dengan AES
        encoded_sensitive_data = self.encrypt_metadata(sensitive_data, \
                                                       encryption_key)

        # Mengubah public_key menjadi string dengan format \
        # {"X": ..., "Y": ...}
        public_key_str = f'({{"X": {public_key[0]}, \
            "Y": {public_key[1]}}})'

        # Membuka PDF asli
        with pikepdf.open(self.pdf_path) as pdf:
            # Menyimpan metadata yang terlihat (r, s) dan public_key
            pdf.docinfo['/Signature'] = f'R={r}, S={s}, \
                PublicKey={public_key_str}'
            pdf.docinfo['/SignatureType'] = 'ECDSA'
            pdf.docinfo['/SignatureDate'] = datetime.now().strftime\
                ('%Y-%m-%d')

            # Menyimpan data terenkripsi sebagai metadata tambahan
            pdf.docinfo['/CurveParams'] = encoded_sensitive_data

            pdf.docinfo['/EncryptionKey'] = encryption_key_encoded
            # Debug: Menampilkan metadata yang tersimpan
            print("Metadata yang disimpan:")
            for key, value in pdf.docinfo.items():
                print(f"{key}: {value}")  # Debug: tampilkan semua metadata

            # Menentukan jalur file PDF yang ditandatangani
            signed_pdf_path = self.pdf_path.replace(".pdf", "_signed.pdf")
            print(f"Menyimpan file ditandatangani sebagai: {signed_pdf_path}")
            # Menyimpan PDF yang ditandatangani dengan metadata
            pdf.save(signed_pdf_path)

        # Memastikan bahwa file telah disimpan
        if os.path.exists(signed_pdf_path):
            self.signature_area.append(f"\nFile berhasil ditandatangani dan \
                                       disimpan sebagai: {signed_pdf_path}")
            self.signed_pdf_path = signed_pdf_path
            print(f"File berhasil disimpan: {signed_pdf_path}")

        else:
            self.signature_area.setText("Gagal menyimpan file \
                                        yang ditandatangani.")
            print("File tidak ditemukan setelah penyimpanan.")

    except Exception as e:
        self.signature_area.setText(f"Gagal menyimpan metadata \
                                    tanda tangan: {e}")
        print(f"Kesalahan: {e}")  # Debug: cetak kesalahan
```

```
def download_signed_pdf(self):
    if hasattr(self, 'signed_pdf_path'):
        self.signature_area.append(f"Download berhasil: \
                                   {self.signed_pdf_path}")
    else:
        self.signature_area.append\
            ("Belum ada file yang ditandatangani.")

def generate_random_k(self, n):
    """Menghasilkan nonce k yang acak dan valid."""
    k = random.randint(1, n - 1)
    return k

def point_multiply(self, k, P, a, b, p, n):
    """Menghitung hasil kali titik P dengan
        skalar k pada kurva eliptik."""
    if k <= 0 or k >= n:
        raise ValueError(f"Skalar k = {k} tidak valid untuk kurva.")

    result = None  # Titik tak hingga
    addend = P

    # Pastikan P valid sebelum memulai perhitungan
    if not self.is_point_on_curve(addend[0], addend[1], a, b, p):
        raise ValueError(f"Titik awal {addend} tidak berada pada kurva.")

    while k:
        if k & 1:  # Jika k adalah ganjil
            result = self.point_add(result, addend, a, b, p)
        addend = self.point_add(addend, addend, a, b, p)
        k >>= 1  # Pembagian k dengan 2
    return result
def point_add(self, P, Q, a, b, p):
    """Menambahkan dua titik P dan Q pada kurva eliptik."""
    if P is None:
        return Q
    if Q is None:
        return

    x1, y1 = P
    x2, y2 = Q

    # Memastikan titik-titik berada di kurva sebelum melanjutkan
    if not self.is_point_on_curve(x1, y1, a, b, p):
        raise ValueError(f"Titik {P} tidak valid pada kurva.")
    if not self.is_point_on_curve(x2, y2, a, b, p):
        raise ValueError(f"Titik {Q} tidak valid pada kurva.")

    if P != Q:
        # Hitung kemiringan (slope)
        m = (y2 - y1) * pow(x2 - x1, -1, p) % p
    else:
        # Kasus P = Q (gunakan rumus untuk dua kali lipat)
        m = (3 * x1 ** 2 + a) * pow(2 * y1, -1, p) % p

    x3 = (m ** 2 - x1 - x2) % p
    y3 = (m * (x1 - x3) - y1) % p

    return (x3, y3)
```

```
    def is_point_on_curve(self, x, y, a, b, p):
        """Memeriksa apakah titik (x, y)
        berada pada kurva eliptik y^2 ≡ x^3 + ax + b (mod p)."""
        if x < 0 or x >= p or y < 0 or y >= p:
            raise ValueError(f"Titik ({x}, {y}) berada di luar batas kurva.")
        return (y * y) % p == (x * x * x + a * x + b) % p


class VerificationWindow(QWidget):
    def __init__(self, main_window):
        super().__init__()
        self.main_window = main_window
        self.setWindowTitle("Verifikasi Tanda Tangan Digital")

        # Layout utama
        self.layout = QVBoxLayout()
        self.layout.setContentsMargins(20, 10, 20, 10)
        self.layout.setSpacing(10)

        # Mengatur ukuran jendela agar tetap dan tidak bisa diperbesar
        self.setFixedSize(700, 800)

        # Title label dengan font yang lebih besar dan bold
        self.title_label = QLabel("Verifikasi Tanda Tangan Digital")
        self.title_label.setFont(QFont('Arial', 17, QFont.Bold))
        self.title_label.setAlignment(Qt.AlignCenter)
        self.layout.addWidget(self.title_label, alignment=Qt.AlignCenter)

        # Group box untuk input pesan (PDF)
        self.message_group_box = QGroupBox("Pilih dan Hash Pesan (PDF)")
        self.message_group_box.setFont(QFont('Arial', 11, QFont.Bold))
        self.message_group_box_layout = QVBoxLayout()

        # Label "Pesan (PDF)" dengan font yang lebih besar
        self.message_label = QLabel("Pesan (PDF):")
        self.message_label.setFont(QFont('Arial', 10))
        self.message_input = QLineEdit()
        self.message_browse_button = QPushButton("Pilih PDF")
        self.style_button(self.message_browse_button)

        # Layout tombol "Pilih PDF" di tengah
        self.message_group_box_layout.addWidget(self.message_label)
        self.message_group_box_layout.addWidget(self.message_input)
        self.message_group_box_layout.addWidget(self.message_browse_button, \
                                                alignment=Qt.AlignCenter)

        # Tombol Hash di tengah
        self.hash_button = QPushButton("Hash Pesan")
        self.style_button(self.hash_button)
        self.message_group_box_layout.addWidget(self.hash_button,\
                                                alignment=Qt.AlignCenter)
        self.message_browse_button.clicked.connect(self.browse_pdf_file)
        self.message_group_box.setLayout(self.message_group_box_layout)
        self.layout.addWidget(self.message_group_box)
```

```
# Hasil Message Digest
self.digest_result = QTextEdit("Hasil Message Digest akan \
                               muncul di sini.")
self.digest_result.setFont(QFont('Arial', 10))
self.digest_result.setReadOnly(True)
self.digest_result.setFixedHeight(100)  # Menambah tinggi
self.hash_button.clicked.connect(self.hash_message)
self.layout.addWidget(self.digest_result)

# Group box untuk input r, s, dan kunci publik
self.signature_group_box = QGroupBox("Masukkan Nilai Tanda \
                                     Tangan Digital")
self.signature_group_box.setFont(QFont('Arial', 11, QFont.Bold))
self.signature_group_box_layout = QFormLayout()

# r input
self.r_label = QLabel("Nilai r:")
self.r_label.setFont(QFont('Arial', 10, QFont.Bold))
self.r_input = QLineEdit()
self.style_input(self.r_input)
self.signature_group_box_layout.addRow(self.r_label, self.r_input)

# s input
self.s_label = QLabel("Nilai s:")
self.s_label.setFont(QFont('Arial', 10, QFont.Bold))
self.s_input = QLineEdit()
self.style_input(self.s_input)
self.signature_group_box_layout.addRow(self.s_label, self.s_input)

# Kunci Publik X input
self.public_key_x_label = QLabel("Kunci Publik x:")
self.public_key_x_label.setFont(QFont('Arial', 10, QFont.Bold))
self.public_key_x_input = QLineEdit()
self.style_input(self.public_key_x_input)
self.signature_group_box_layout.addRow(self.public_key_x_label, \
                                       self.public_key_x_input)

# Kunci Publik Y input
self.public_key_y_label = QLabel("Kunci Publik y:")
self.public_key_y_label.setFont(QFont('Arial', 10, QFont.Bold))
self.public_key_y_input = QLineEdit()
self.style_input(self.public_key_y_input)
self.signature_group_box_layout.addRow(self.public_key_y_label,\
                                       self.public_key_y_input)

self.signature_group_box.setLayout(self.signature_group_box_layout)
self.layout.addWidget(self.signature_group_box)

# Hasil Verifikasi
self.verification_result = QTextEdit\
    ("Hasil verifikasi akan muncul di sini.")
self.verification_result.setFont(QFont('Arial', 10))
self.verification_result.setReadOnly(True)
self.verification_result.setFixedHeight(100)
self.layout.addWidget(self.verification_result)

# Tombol Verifikasi
self.verify_button = QPushButton("Verifikasi")
self.verify_button.setFixedSize(250, 50)
self.style_button(self.verify_button)
self.layout.addWidget(self.verify_button, alignment=Qt.AlignCenter)
```

```
    self.setLayout(self.layout)
    self.verify_button.clicked.connect(self.verify_signature)

def style_input(self, input_field):
    input_field.setStyleSheet("""
        QLineEdit {
            border: 1px solid #A9A9A9;
            border-radius: 5px;
            padding: 5px;
            background: qlineargradient(spread:pad, x1:0, y1:0, \
                        x2:1, y2:1, stop:0 #FFFFFF, stop:1 #F0F0F0);
            font-size: 14px;
        }
        QLineEdit:focus {
            border: 1px solid #4CAF50;
            background: qlineargradient(spread:pad, x1:0, y1:0, x2:1, y2:1,\
                        stop:0 #FFFFFF, stop:1 #E0F7E0);
        }
    """)

def style_button(self, button):
    button.setStyleSheet("""
        QPushButton {
            background-color: #4CAF50;
            color: white;
            font-weight: bold;
            font-size: 15px;
            border-radius: 5px;
            padding: 10px 20px;
        }
        QPushButton:hover {
            background-color: #45A049;
        }
    """)
    button.setFixedWidth(180)

def browse_pdf_file(self):
    options = QFileDialog.Options()
    file_path, _ = QFileDialog.getOpenFileName(self, "Pilih File PDF", "",\
                                               "PDF Files (*.pdf);;\
                                               All Files (*)", options=options)
    if file_path:
        self.message_input.setText(file_path)
        self.read_signature_metadata(file_path)

def read_signature_metadata(self, pdf_path):
    try:
        with open(pdf_path, "rb") as pdf_file:
            pdf_reader = PdfReader(pdf_file)
            metadata = pdf_reader.metadata

            if metadata:
                signature = metadata.get('/Signature', '')
                curve_params_encrypted = metadata.get\
                    ('/CurveParams', '')
                encryption_key_encoded = metadata.get\
                    ('/EncryptionKey', '')

                if not encryption_key_encoded:
                    self.verification_result.setText\
                    ("Kunci enkripsi tidak ditemukan dalam metadata PDF.")
                    return
```

```
                encryption_key = base64.b64decode(encryption_key_encoded)

                if signature:
                    self.parse_metadata(signature)

                if curve_params_encrypted:
                    decrypted_curve_params = self.decrypt_signature\
                        (curve_params_encrypted, encryption_key)

                    if decrypted_curve_params:
                        print(f"Parameter kurva setelah Dekripsi: \
                            {decrypted_curve_params}")
                        params = dict(item.split("=") for item\
                                in decrypted_curve_params.split(";"))

                        # Simpan hasil dekripsi dalam variabel kelas\
                        #  untuk digunakan selanjutnya
                        self.a = int(params.get('a', ''))
                        self.b = int(params.get('b', ''))
                        self.p = int(params.get('p', ''))
                        self.G = tuple(map(int, params.get('G', '(0, 0)')\
                                .strip('()').split(',')))
                        self.n = int(params.get('n', ''))
                        self.k = int(params.get('k', ''))

                        # Debug: Tampilkan nilai yang telah disimpan
                        print(f"Parameter kurva yang telah disimpan: a={self.a},\
                            b={self.b}, p={self.p}, G={self.G}, n={self.n},\
                              k={self.k}")

                        # Tidak menampilkan di UI, hanya untuk perhitungan
                        self.verification_result.setText\
                        ("Metadata tanda tangan berhasil didekripsi dan dibaca.")
                    else:
                        self.verification_result.setText\
                            ("Gagal mendekripsi parameter kurva.")
                else:
                    self.verification_result.setText\
                    ("Tidak ada data parameter kurva yang ditemukan dalam metadata.")
            else:
                self.verification_result.setText\
                ("Metadata tidak tersedia dalam file PDF.")
    except Exception as e:
        self.verification_result.setText\
            (f"Kesalahan dalam membaca metadata: {e}")

def parse_metadata(self, signature):
    try:
        # Debug metadata awal
        print(f"Signature metadata yang diterima: {signature}")

        # Hapus spasi ekstra
        signature = signature.strip()

        # Gunakan regex untuk menangkap R, S, dan \
        # PublicKey secara langsung
        r_match = re.search(r'R=(\d+)', signature)
        s_match = re.search(r'S=(\d+)', signature)
        public_key_match = re.search\
            (r'PublicKey=\(\{"X":\s*(\d+),\s*"Y":\s*(\d+)\}\)', signature)

        found_r, found_s, found_public_key = False, False, False

        # Mengatur nilai R jika ditemukan
        if r_match:
            r_value = r_match.group(1)
            print(f"Nilai R ditemukan: {r_value}")  # Debug output
            self.r_input.setText(r_value)
            found_r = True
```

```
            # Mengatur nilai S jika ditemukan
            if s_match:
                s_value = s_match.group(1)
                print(f"Nilai S ditemukan: {s_value}")  # Debug output
                self.s_input.setText(s_value)
                found_s = True

            # Mengatur nilai PublicKey jika ditemukan
            if public_key_match:
                pk_x = public_key_match.group(1)
                pk_y = public_key_match.group(2)
                print(f"Nilai PublicKey X ditemukan: {pk_x}")  # Debug output
                print(f"Nilai PublicKey Y ditemukan: {pk_y}")  # Debug output
                self.public_key_x_input.setText(pk_x)
                self.public_key_y_input.setText(pk_y)
                found_public_key = True

            # Periksa jika semua komponen ditemukan
            if found_r and found_s and found_public_key:
                self.verification_result.setText\
                    ("Semua metadata berhasil dibaca.")
            else:
                missing_fields = []
                if not found_r:
                    missing_fields.append("R")
                if not found_s:
                    missing_fields.append("S")
                if not found_public_key:
                    missing_fields.append("PublicKey")
                self.verification_result.setText\
                    (f"Metadata tidak lengkap. Tidak ditemukan: \
                     {', '.join(missing_fields)}.")
        except Exception as e:
            print(f"Kesalahan dalam pemrosesan metadata: {e}")
            self.verification_result.setText\
                (f"Kesalahan dalam pemrosesan metadata: {e}")

    def decrypt_signature(self, encrypted_signature: str, key: bytes):
        """Mendekripsi data tanda tangan yang dienkripsi dengan AES-CBC"""
        try:
            encrypted_data_with_iv = base64.b64decode(encrypted_signature)

            if len(encrypted_data_with_iv) < 16:
                raise ValueError\
                ("Data terenkripsi kurang dari panjang IV yang diperlukan.")

            iv = encrypted_data_with_iv[:16]  # IV
            encrypted_data = encrypted_data_with_iv[16:]

            cipher = AES.new(key, AES.MODE_CBC, iv)
            decrypted_data = unpad(cipher.decrypt(encrypted_data),\
                                   AES.block_size).decode("utf-8")

            return decrypted_data
        except Exception as e:
            print(f"Kesalahan dalam dekripsi: {e}")
            return ""

    def hash_message(self):
        try:
            pdf_path = self.message_input.text()
            if not pdf_path:
                self.digest_result.setText("Pilih file PDF terlebih dahulu.")
                return
```

```
        with open(pdf_path, "rb") as pdf_file:
            pdf_reader = PdfReader(pdf_file)
            text_content = "".join(page.extract_text() or "" \
                                   for page in pdf_reader.pages)

        hashed_message = hashlib.sha256(text_content.encode())\
            .hexdigest()
        self.digest_result.setText(f"Message Digest (SHA-256):\
                                   {hashed_message}")
        self.hashed_message = hashed_message

    except Exception as e:
        self.digest_result.setText\
            (f"Kesalahan dalam hashing file PDF: {e}")


def is_point_on_curve(self, x, y, a, b, p):
    """Memeriksa apakah titik (x, y)
    berada pada kurva eliptik y^2 ≡ x^3 + ax + b (mod p)."""
    if x < 0 or x >= p or y < 0 or y >= p:
        raise ValueError(f"Titik ({x}, {y}) \
                         berada di luar batas kurva.")
    return (y * y) % p == (x * x * x + a * x + b) % p

def point_multiplication(self, scalar, P, a, b, p, n):
    """Menghitung hasil kali titik P dengan skalar pada \
        kurva eliptik."""
    print(f"Nilai scalar yang diterima untuk point \
          multiplication: {scalar}")

    # Validasi nilai scalar
    if scalar <= 0 or scalar >= n:
        raise ValueError(f"Nilai scalar = {scalar} \
                         tidak valid untuk kurva.")
```

```
    result = None  # Titik identitas (titik nol)
    addend = P

    # Pastikan P valid sebelum memulai perhitungan
    if not self.is_point_on_curve(addend[0], addend[1], a, b, p):
        raise ValueError(f"Titik awal {addend} tidak berada pada kurva.")

    print(f"Point P valid: {P}")

    while scalar:
        print(f"Nilai scalar saat ini: {scalar}, \
              bit terkecil: {scalar & 1}")
        if scalar & 1:  # Jika scalar adalah ganjil
            result = self.point_addition(result, addend, a, b, p)
            if result is None:
                print("Hasil point addition menghasilkan titik\
                      identitas (None). Verifikasi gagal.")
                return None
        addend = self.point_addition(addend, addend, a, b, p)
        if addend is None:
            print("Hasil point addition pada langkah perkalian \
                  titik menghasilkan titik identitas (None).")
            return None  # Hentikan proses jika hasil addend adalah None
        scalar >>= 1  # Pembagian scalar dengan 2

    return result
```

```
def point_addition(self, P, Q, a, b, p):
    """Menambahkan dua titik pada kurva eliptik."""
    # Penanganan titik identitas
    if P is None:
        print(f"Titik P adalah titik identitas.")
        return Q
    if Q is None:
        print(f"Titik Q adalah titik identitas.")
        return P

    x1, y1 = P
    x2, y2 = Q

    # Jika titik sama
    if x1 == x2 and y1 == y2:
        if y1 == 0:  # Menangani titik identitas untuk titik sama
            print("Titik identitas ditemukan saat titik sama.")
            return None  # Kembalikan None sebagai titik identitas
        m = (3 * x1 * x1 + a) * pow(2 * y1, -1, p) % p
    elif x1 == x2 and y1 != y2:  # Kasus khusus jika x1 == x2 \
        #tetapi y1 != y2
        print(f"Titik x1 dan x2 sama tetapi y1 dan y2 berbeda. \
              Menghasilkan titik identitas.")
        return None  # Kembalikan None sebagai titik identitas
    else:
        m = (y2 - y1) * pow(x2 - x1, -1, p) % p

    x3 = (m * m - x1 - x2) % p
    y3 = (m * (x1 - x3) - y1) % p

    # Debugging nilai sebelum mod dengan curve_order
    print(f"m: {m}, x3 sebelum mod: {m * m - x1 - x2}, \
          y3 sebelum mod: {m * (x1 - x3) - y1}")
    # Memastikan titik yang dihitung valid
    if x3 is None or y3 is None:
        raise ValueError("Hasil penambahan titik tidak valid.")

    return (x3, y3)

def verify_signature(self):
    try:
        # Ambil path file PDF terlebih dahulu
        pdf_path = self.message_input.text()
        if not pdf_path:
            self.verification_result.setText\
                ("Pilih file PDF terlebih dahulu.")
            return

        # Cek apakah ada message yang dihash
        if not hasattr(self.main_window, 'hashed_message'):
            self.verification_result.setText\
                ("Hash pesan terlebih dahulu sebelum menandatangani.")
            return

        # Ambil nilai r, s, dan kunci publik
        r = int(self.r_input.text())
        s = int(self.s_input.text())
        pub_key_x = int(self.public_key_x_input.text())
        pub_key_y = int(self.public_key_y_input.text())

        # Tampilkan hasil verifikasi untuk r, s
        self.verification_result.append(f"r: {r}, s: {s}")
```

```
# Pastikan parameter kurva sudah disimpan sebelumnya
if not all(hasattr(self, param) \
        for param in ['a', 'b', 'p', 'G', 'n']):
    self.verification_result.setText\
        ("Parameter kurva belum dimuat atau tidak lengkap.")
    return

# Ambil parameter kurva dari atribut telah disimpan sebelumnya
a = self.a
b = self.b
p = self.p
G = self.G
curve_order = self.n
k = self.k

# Log parameter kurva untuk debug
print(f"Parameter kurva: a={a}, b={b}, p={p}, \
    G={G}, curve_order={curve_order}, k={k}")


if a is None or b is None or p is None or not G\
    or curve_order <= 0:
    self.verification_result.setText\
        ("Gagal memuat parameter dari metadata PDF\
        atau parameter tidak valid.")
    return

# Cek keberadaan titik G dan order kurva
if not G or curve_order <= 0:
    self.verification_result.setText\
        ("Informasi titik G atau order kurva tidak valid.")
    return
# Verifikasi nilai r, s
if not (1 <= r < curve_order) or not (1 <= s < curve_order):
    self.verification_result.setText\
        ("Nilai r dan s tidak valid. Harus antara [1, n-1].")
    return

# Memastikan kunci publik valid pada kurva
if not self.is_point_on_curve(pub_key_x, pub_key_y, a, b, p):
    self.verification_result.setText\
        ("Kunci publik tidak valid pada kurva yang ditentukan.")
    return

# Memproses hashing pesan asli
with open(pdf_path, "rb") as pdf_file:
    pdf_reader = PdfReader(pdf_file)
    text_content = "".join(page.extract_text() or "" \
                           for page in pdf_reader.pages)

hashed_message = hashlib.sha256(text_content.encode()).hexdigest()

# Memverifikasi tanda tangan
w = pow(s, -1, curve_order)
u1 = (int(hashed_message, 16) * w) % curve_order
u2 = (r * w) % curve_order

# Debug nilai u1 dan u2
print(f"Nilai u1: {u1}, u2: {u2}")

# Hitung titik pada kurva
x1, y1 = self.point_multiplication(u1, G, a, b, p, curve_order)
x2, y2 = self.point_multiplication\
    (u2, (pub_key_x, pub_key_y), a, b, p, curve_order)
```

```
            # Debug hasil perkalian titik
            print(f"Hasil u1 * G: ({x1}, {y1})")
            print(f"Hasil u2 * Public Key: ({x2}, {y2})")

            if x1 is None or x2 is None:
                self.verification_result.setText\
                    ("Kesalahan dalam perkalian titik.")
                return

            # Penambahan titik pada kurva
            x3, y3 = self.point_addition((x1, y1), (x2, y2), a, b, p)

            # Debug hasil penambahan titik
            print(f"Hasil penambahan titik sebelum mod: ({x3}, {y3})")

            if x3 is None:
                self.verification_result.setText\
                    ("Kesalahan dalam penambahan titik.")
                return

            # Lakukan modulus dengan curve order (n)
            v = x3 % curve_order
            z = y3 % curve_order

            # Debug hasil setelah mod
            print(f"Hasil setelah mod dengan curve_order \
                  ({curve_order}): ({v}, {z})")

            if v == r:
                self.verification_result.setText("Tanda tangan valid.")
            else:
                self.verification_result.setText("Tanda tangan tidak valid.")
        except ValueError as e:
            self.verification_result.setText(f"Kesalahan nilai: {e}")
        except Exception as e:
            self.verification_result.setText\
                (f"Kesalahan dalam verifikasi: {e}")


# Main Window setup
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Aplikasi Tanda Tangan Digital")
        self.setFixedSize(700, 900)

        self.tab_widget = QTabWidget()
        self.equation = ""
        self.points = []
        self.hashed_message = ""

        # Setel font tab (label pada tab)
        self.tab_widget.setStyleSheet("""
            QTabBar::tab {
                font-size: 7pt;
                padding: 8px;
            }
            QTabBar::tab:selected {
                background-color: #4CAF50;
                color: white;
            }
        """)

        self.curve_points_window = CurvePointsWindow(self)
        self.signing_window = SigningWindow(self)
        self.verification_window = VerificationWindow(self)
```

```
        # Menambahkan tab
        self.tab_widget.addTab(self.curve_points_window, "Titik Kurva")
        self.tab_widget.addTab(KeyGenerationWindow\
                               (self, self.curve_points_window), "Generate Key")
        self.tab_widget.addTab(self.signing_window, "Signing")
        self.tab_widget.addTab(self.verification_window, "Verification")

        self.setCentralWidget(self.tab_widget)

    def set_equation(self, equation, points=None):
        self.equation = equation
        if points is not None:
            self.points = points

    def get_curve_params(self):
        parts = self.equation.split("mod")
        equation_part = parts[0].strip()
        equation_terms = equation_part.split("+")
        a = int(equation_terms[1].replace("x", "").strip())
        b = int(equation_terms[2].strip())
        p = int(parts[1].strip())
        return a, b, p

if __name__ == "__main__":
    app = QApplication(sys.argv)
    main_window = MainWindow()
    main_window.show()
    sys.exit(app.exec_())
```

RIWAYAT HIDUP

Zakiyah Nur Millah, lahir di Pasuruan 29 Februari 2004 sebagai anak kedua dari dua bersaudara. Pendidikan dasar hingga menengah pertama ditempuh di Yayasan Bani Chozin Elkafa, mencakup pendidikan RA, Madrasah Ibtidaiyah (MI), Madrasah Tsanawiyah (MTs).

Sejak masa Madrasah Ibtidaiyah (MI) hingga MTs, penulis aktif dalam berbagai organisasi serta kegiatan pengembangan minat dan bakat, yang membawanya berkesempatan mewakili sekolah dalam berbagai kompetisi. Pada tahun 2018, penulis melanjutkan pendidikan di SMA Excellent Al-Yasini sekaligus menjalani kehidupan pesantren selama tiga tahun. Selama masa SMA, penulis tetap aktif dalam organisasi serta pengembangan minat dan bakat, bahkan turut mengharumkan nama sekolah melalui berbagai ajang kompetisi dengan sekolah lain.

Setelah lulus pada tahun 2021, penulis diterima di Universitas Islam Negeri Maulana Malik Ibrahim Malang melalui jalur SNMPTN pada Program Studi Matematika, Fakultas Sains dan Teknologi. Selama perkuliahan, penulis mengabdikan diri sebagai musyrifah di Ma'had Sunan Ampel Al-Aly (MSAA) UIN Malang selama dua tahun dan tergabung dalam Divisi Kesantrian, yang mengelola berbagai kegiatan keagamaan serta pembinaan santri. Selain itu, penulis juga aktif dalam berbagai organisasi kemahasiswaan, termasuk menjadi anggota Unit Kegiatan Mahasiswa (UKM) Seni Religi (SR) serta menjabat sebagai pengurus di Unit Pengembangan Kegiatan Mahasiswa (UPKM) JDFI**.**

Dengan berbagai pengalaman dan pengabdian yang telah dijalani, penulis terus berupaya mengembangkan diri serta berkontribusi dalam lingkungan akademik dan sosial. Untuk informasi lebih lanjut atau diskusi terkait penelitian ini, dapat menghubungi penulis melalui email: zakiyahnur050117@gmail.com.

KEMENTERIAN AGAMA RI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Gajayana No.50 Dinoyo Malang Telp. / Fax. (0341)558933

BUKTI KONSULTASI SKRIPSI

Nama : Zakiyah Nur Millah
NIM : 210601110029
Fakultas / Jurusan : Sains dan Teknologi / Matematika
Judul Skripsi : Implementasi Fungsi *Hash* SHA-256 dan Kriptografi Algoritma ECDSA (*Elliptic Curve Digital Signature Algorithm)* dalam Pembuatan Tanda Tangan Digital
Pembimbing I : Muhammad Khudzaifah, M.Si
Pembimbing II : Erna Herawati, M.Pd.

No	Tanggal	Hal	Tanda Tangan
1.	28 Mei 2024	Konsultasi Topik dan Data	1.
2.	17 September 2024	Konsultasi Bab I, II, dan III	2.
3.	26 September 2024	Konsultasi Kajian Agama Bab I dan II	3.
4.	02 Oktober 2024	Konsultasi Kajian Agama Bab I dan II	4.
5.	04 Oktober 2024	ACC Kajian Agama Bab I dan II	5.
6.	08 Oktober 2024	ACC Bab I, II, dan III	6.
7.	10 Oktober 2024	ACC Seminar Proposal	7.
8.	07 November 2024	Konsultasi Revisi Seminar Proposal	8.
9.	08 November 2024	Konsultasi Bab IV, dan V	9.
10.	11 November 2024	Konsultasi Kajian Agama Bab IV	10.
11.	12 November 2024	ACC Kajian Agama Bab IV	11.
12.	13 November 2024	ACC Seminar Hasil	12.
13.	4 Maret 2025	Konsultasi Revisi Seminar Hasil	13.
14.	12 Maret 2025	ACC Sidang Skripsi	14.

KEMENTERIAN AGAMA RI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Gajayana No.50 Dinoyo Malang Telp. / Fax. (0341)558933

No	Tanggal	Hal	Tanda Tangan
15.	16 Mei 2025	ACC Keseluruhan	15.

Malang, 16 Mei 2025
Mengetahui,
Ketua Program Studi Matematika

Dr. Elly Susanti, M.Sc.
NIP. 19741129 200012 2 005