DAFTAR PUSTAKA

Arikunto, S. (2011). *Prosedur Penelitian Suatu Pendekatan Praktik* (14 ed.). Rineka Cipta.

Dr Asha Ambhaikar, Mr. A. G. (2021). AES AND RSA-BASED HYBRID ALGORITHMS FOR MESSAGE ENCRYPTION & DECRYPTION. *INFORMATION TECHNOLOGY IN INDUSTRY*, *9*(1), 273–279. https://doi.org/10.17762/itii.v9i1.129

Fauziah, N. A., Rachmawanto, E. H., Moses Setiadi, D. R. I., & Sari, C. A. (2018). Design and Implementation of AES and SHA-256 Cryptography for Securing Multimedia File over Android Chat Application. *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 146–151. https://doi.org/10.1109/ISRITI.2018.8864485

G. Chaloop, S., & Z. Abdullah, M. (2022). ENHANCING HYBRID SECURITY APPROACH USING AES AND RSA ALGORITHMS. *Journal of Engineering and Sustainable Development*, *25*(4), 58–66. https://doi.org/10.31272/jeasd.25.4.6

Hagos, T. (2018). Learn Android Studio 3 with Kotlin. Dalam *Learn Android Studio 3 with Kotlin*. Apress. https://doi.org/10.1007/978-1-4842-3907-0

Hermawan, A., Iman, E., & Ujianto, H. (2021). Implementasi Enkripsi Data Menggunakan Kombinasi AES dan RSA. *InfoTekJar: Jurnal Nasional Informatika Dan Teknologi Jaringan*, *5*(2), 325–330. https://doi.org/10.30743/infotekjar.v5i2.3585

Katz, J., & Lindell, Y. (2020). *Introduction to Modern Cryptography*. Chapman and Hall/CRC. https://doi.org/10.1201/9781351133036

Munir, R. (2019). *Kriptografi* (Edisi Kedua). Penerbit Informatika.

Nasution, R. N., & Triandi, B. (2021). IMPLEMENTASI METODE RSA DAN AES UNTUK MENGAMANKAN FILE WINRAR DAN ZIP. *IT (INFORMATIC TECHNIQUE) JOURNAL*, *8*(1), 42–53. https://doi.org/10.22303/it.8.1.2020.42-53

Paar, C., & Pelzl, J. (2010). *Understanding Cryptography*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-04101-3

Pasaribu, H., Sitanggang, D., Damanik, R. R., & Rudianto Sitompul, A. C. (2018). Combination of advanced encryption standard 256 bits with md5 to secure documents on android smartphone. *Journal of Physics: Conference Series*, *1007*(1), 012014. https://doi.org/10.1088/1742-6596/1007/1/012014

Pattanavichai, S. (2022). Program for Simulation and Testing of Apply Cryptography of Advance Encryption Standard (AES) Algorithm with Rivest-Shamir-Adleman (RSA) Algorithm for Good Performance. *International Journal of Electronics and Telecommunications*, *68*(3), 475–481. https://doi.org/10.24425/ijet.2022.141263

Ramadani, S., & Sauda, S. (2020). Penerapan Algoritma AES dan DSA Menggunakan Hybrid Cryptosystem untuk Keamanan Data. *JURIKOM (Jurnal Riset Komputer)*, *7*(4), 523–529. https://doi.org/10.30865/jurikom.v7i4.2055

Sadikin, M. A., & Wardhani, R. W. (2016). Implementation of RSA 2048-bit and AES 256-bit with digital signature for secure electronic health record application. *2016 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, *10*(2), 387–392. https://doi.org/10.1109/ISITIA.2016.7828691

Siringoringo, R. (2020). Analisis dan Implementasi Algoritma Rijndael (AES) dan Kriptografi RSA pada Pengamanan File. *KAKIFIKOM (Kumpulan Artikel Karya Ilmiah Fakultas Ilmu Komputer)*, *02*(01), 31–42. https://doi.org/10.54367/kakifikom.v2i1.666

Stallings, W. (2017). *Cryptography and network security : principles and practice*.

Sugiyono. (2018). *Metode penelitian kuantitatif, kualitatif dan kombinasi (Mixed Methods)* (Sutopo, Ed.). Alfabeta.

Zed, M. (2004). *Metode Penelitian Kepustakaan*. Yayasan Pustaka Obor Indonesia.

LAMPIRAN

Lampiran 1. Proses Enkripsi AES 256 bit

R	State awal				Setelah SubBytes				Setelah ShiftRows				Setelah MixColumns				Setelah AddRoundKey			
1	00	40	80	c0	63	09	cd	ba	63	09	cd	ba	5f	57	f7	1d	4f	43	ef	01
	10	50	90	d0	ca	53	60	70	53	60	70	ca	72	f5	be	b9	63	e0	a7	a4
	20	60	a0	e0	b7	d0	e0	e1	e0	e1	b7	d0	64	bc	3b	f9	76	aa	21	e7
	30	70	b0	f0	04	51	e7	8c	8c	04	51	e7	15	92	29	1a	06	85	32	05
2	4f	43	ef	01	84	1a	df	7c	84	1a	df	7c	bd	2b	d1	61	18	8a	78	c4
	63	e0	a7	a4	fb	e1	5c	49	e1	5c	49	fb	2a	6a	92	5d	59	1c	ed	2f
	76	aa	21	e7	38	ac	fd	94	fd	94	38	ac	39	c4	44	a1	fb	00	8a	61
	06	85	32	05	6f	97	23	6b	6b	6f	97	23	5d	38	3e	95	c2	a0	ad	09
3	18	8a	78	c4	ad	7e	bc	1c	ad	7e	bc	1c	81	c9	b3	88	97	cb	a9	8e
	59	1c	ed	2f	cb	9c	55	15	9c	55	15	cb	0d	db	67	a1	5c	9f	3a	e1
	fb	00	8a	61	0f	63	7e	ef	7e	ef	0f	63	ce	81	8c	b5	66	3f	28	0f
	c2	a0	ad	09	25	e0	95	01	01	25	e0	95	0c	72	1e	bd	c1	a8	df	63
4	97	cb	a9	8e	88	1f	d3	19	88	1f	d3	19	b2	ab	5f	07	1c	a4	f9	04
	5c	9f	3a	e1	4a	db	80	f8	db	80	f8	4a	82	e6	af	8c	05	17	21	07
	66	3f	28	0f	33	75	34	76	34	76	33	75	2d	fb	10	00	f2	e0	c5	15
	c1	a8	df	63	78	c2	9e	fb	fb	78	c2	9e	81	27	3a	33	71	4f	c1	54
5	1c	a4	f9	04	9c	49	99	f2	9c	49	99	f2	ae	74	d7	db	c3	1b	a2	a8
	05	17	21	07	6b	f0	fd	51	f0	fd	41	6b	b6	e0	3f	64	57	45	c7	dc
	f2	e0	c5	15	89	e1	a6	59	a6	59	89	e1	5b	f8	56	c8	aa	b7	bd	99
	71	4f	c1	54	a3	84	78	20	20	a3	84	78	a9	22	7b	77	e1	b0	28	fa
6	c3	1b	a2	a8	2e	af	3a	c2	2e	af	3a	c2	b9	02	ae	ef	7f	cb	c1	83
	57	45	c7	dc	5b	6e	c6	86	6e	c6	86	5b	51	e9	25	a0	07	4e	0c	75
	aa	b7	bd	99	ac	a9	7a	ee	7a	ee	ac	a9	c3	bd	cd	8c	41	24	81	d5
	e1	b0	28	fa	f8	e7	34	2d	2d	f8	e7	34	3c	29	b1	c7	43	3e	5d	4c
	7f	cb	c1	83	d2	1f	78	ec	d2	1f	78	ec	eb	3e	7d	ed	d6	6c	5a	b9
	07	4e	0c	75	c5	2f	fe	9d	2f	fe	9d	c5	b1	e7	75	6b	53	a0	ca	6c

7	41	24	81	d5	83	36	0c	03	0c	03	83	36	9e	c9	35	91	a4	bc	ab	5e
	43	3e	5d	4c	1a	b2	4c	29	29	1a	b2	4c	1c	e8	e9	44	69	0f	5d	7d
8	d6	6c	5a	b9	f6	50	be	56	f6	50	be	56	51	9d	a8	a9	5a	5f	05	68
	53	a0	ca	6c	ed	e0	74	50	e0	74	50	ed	74	a9	b3	74	a8	d2	e1	f3
	a4	bc	ab	5e	49	65	62	58	62	58	49	65	c8	84	e6	a5	58	8d	a3	b9
	69	0f	5d	7d	f9	76	4c	ff	ff	f9	76	4c	66	35	2c	ea	39	7d	88	c5
9	5a	5f	05	68	be	cf	6b	45	be	cf	6b	45	0f	d2	54	4e	4a	c5	64	2a
	a8	d2	e1	f3	c2	b5	f8	0d	b5	f8	0d	c2	77	cc	30	f9	82	7e	3d	f3
	58	8d	a3	b9	6a	5d	0a	56	0a	56	6a	5d	ee	ad	a8	6a	48	7e	e5	e8
	39	7d	88	c5	12	ff	c4	a6	a6	12	ff	c4	31	c0	3f	c3	51	47	0c	c9
10	4a	c5	64	2a	d6	a6	43	e5	d6	a6	43	e5	bd	74	63	e9	c1	ca	70	3b
	82	7e	3d	f3	13	f3	27	0d	f3	27	0d	13	86	8f	0f	33	49	3b	e9	52
	48	7e	e5	e8	52	f3	d9	9b	d9	9b	52	f3	f0	c4	11	12	07	3a	aa	b5
	51	47	0c	c9	d1	a0	fe	dd	dd	d1	a0	fe	ea	f4	c1	33	f6	a0	31	ec
11	c1	ca	70	3b	78	74	51	e2	78	74	51	e2	af	5d	87	d5	5f	ba	50	66
	49	3b	e9	52	3b	e2	1e	00	e2	1e	00	3b	86	6e	e5	c8	9c	c6	40	67
	07	3a	aa	b5	c5	80	ac	d5	ac	d5	c5	80	90	1d	fb	90	6a	34	9f	76
	f6	a0	31	ec	42	e0	c7	ce	ce	42	e0	c7	41	d3	ed	13	bf	aa	a7	53
12	5f	ba	50	66	cf	f4	53	33	cf	f4	53	33	74	d8	9c	5b	51	43	14	01
	9c	c6	40	67	de	b4	09	85	b4	09	85	de	27	a6	e8	e0	66	53	fb	92
	6a	34	9f	76	02	18	db	38	db	38	02	18	fa	95	3d	39	04	95	86	25
	bf	aa	a7	53	08	ac	5c	ed	ed	08	ac	5c	e4	26	31	2b	95	03	e4	21
13	51	43	14	01	d1	1a	fa	7c	d1	1a	fa	7c	2c	30	b7	ee	62	99	c9	23
	66	53	fb	92	33	ed	0f	4f	ed	0f	4f	33	21	6f	12	0d	7b	9d	45	f5
	04	95	86	25	f2	2a	44	3f	44	3f	f2	2a	a8	15	c7	a0	ce	5a	ec	6d
	95	03	e4	21	2a	7b	69	fd	fd	2a	7b	69	20	4a	5e	4f	b9	aa	f4	a5
																	Output			
14	62	99	c9	23	aa	ee	dd	26	aa	ee	dd	26					8e	51	ea	4b
	7b	9d	45	f5	21	5e	6e	e6	5e	6e	e6	21					a2	67	fc	49
	ce	5a	ec	6d	8b	be	ce	3c	ce	3c	8b	be					b7	45	49	60
	b9	aa	f4	a5	56	ac	bf	06	06	56	ac	bf					ca	bf	90	89

Lampiran 2. Kode Program AES

```
package com.yogaprasetyo.skripsi.keepsecret.ui.aes

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.core.view.isVisible
import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.google.android.material.snackbar.Snackbar
import com.yogaprasetyo.skripsi.keepsecret.R
import com.yogaprasetyo.skripsi.keepsecret.adapter.ProgressAdapter
import com.yogaprasetyo.skripsi.keepsecret.databinding.DecryptAesOfflineBinding
import com.yogaprasetyo.skripsi.keepsecret.databinding.EncryptAesOfflineBinding
import com.yogaprasetyo.skripsi.keepsecret.databinding.FragmentAesOfflineBinding
import com.yogaprasetyo.skripsi.keepsecret.model.Progress
import com.yogaprasetyo.skripsi.keepsecret.utils.*
import java.io.File
import java.io.InputStream

class AesOfflineFragment : Fragment() {

    private lateinit var outputFileEncrypt: File
    private lateinit var outputFileDecrypt: File

    private lateinit var progressAdapterEncrypt: ProgressAdapter
    private lateinit var progressAdapterDecrypt: ProgressAdapter

    private lateinit var progressOperation: ProgressOperation

    private val questionMark by lazy { getString(R.string.question_mark) }
    private val aesSymmetric by lazy { ECSymmetric() }
    private val progressArray by lazy { ArrayList<Progress>() }

    private val benchmarkTimerEncrypt by lazy {
        BenchmarkTimer("Aes", "Encrypt operation")
    }
    private val benchmarkTimerDecrypt by lazy {
        BenchmarkTimer("Aes", "Decrypt operation")
    }

    private var fis: InputStream? = null
    private var originalSize: String? = null

    private var enc: EncryptAesOfflineBinding? = null
    private var dec: DecryptAesOfflineBinding? = null

    private var _binding: FragmentAesOfflineBinding? = null
    private val binding get() = _binding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
```

```
    _binding = FragmentAesOfflineBinding.inflate(inflater, container, false)
    return binding?.root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    initProperties()
    initRecyclerView()
    initButtonListener()
}

private fun initProperties() {
    enc = binding?.eoperation
    dec = binding?.doperation
    progressAdapterEncrypt = ProgressAdapter()
    progressAdapterDecrypt = ProgressAdapter()
    progressOperation = ProgressOperation(
        progressArray,
        progressAdapterEncrypt,
        progressAdapterDecrypt
    )
}

private fun initRecyclerView() {
    fun configure(rv: RecyclerView): RecyclerView {
        return rv.apply {
            layoutManager = LinearLayoutManager(activity)
            setHasFixedSize(true)
        }
    }

    enc?.rvProgress?.let { configure(it).adapter = progressAdapterEncrypt }
    dec?.rvProgress?.let { configure(it).adapter = progressAdapterDecrypt }
}

private fun initButtonListener() {
    /**
     * Showing operation
     * */
    binding?.btnToggleShowOperation?.toggleButtonListener(
        {
            showCryptoOperation(Operation.ENCRYPT)
            resetAllCacheOrValue()
        },
        {
            showCryptoOperation(Operation.DECRYPT)
            resetAllCacheOrValue()
        }
    )

    /**
     * Encrypt Operation
     * */
    enc?.btnRandomKey?.setOnClickListener { useRandomPassword() }
    enc?.btnUpload?.setOnClickListener { uploadFile(Operation.ENCRYPT) }
    enc?.btnEncrypt?.setOnClickListener { doEncrypt() }

    /**
     * Decrypt Operation
```

```
     * */
    dec?.btnUpload?.setOnClickListener { uploadFile(Operation.DECRYPT) }
    dec?.btnDecrypt?.setOnClickListener { doDecrypt() }
}

private fun showCryptoOperation(operationName: Operation) {
    fun showEncrypt(isVisible: Boolean = true) {
        binding?.eoperation?.root?.isVisible = isVisible
    }

    fun showDecrypt(isVisible: Boolean = true) {
        binding?.doperation?.root?.isVisible = isVisible
    }

    when (operationName) {
        Operation.ENCRYPT -> {
            showEncrypt()
            showDecrypt(false)
        }
        Operation.DECRYPT -> {
            showDecrypt()
            showEncrypt(false)
        }
    }
}

private fun initValueOfView() {
    val chosenFile = getString(R.string.choose_file, questionMark)
    enc?.tvUploadFile?.text = chosenFile
    dec?.tvUploadFile?.text = chosenFile
}

private fun resetAllCacheOrValue() {
    fun clearPropertiesValue() {
        fis = null
        originalSize = null
        progressArray.clear()
        progressAdapterEncrypt.submitList(emptyList())
        progressAdapterDecrypt.submitList(emptyList())
    }

    fun hideOperation() {
        setVisibleStateOfView(
            false,
            arrayListOf(
                enc?.titleProgress,
                enc?.rvProgress,
                enc?.cardResults,
                dec?.titleProgress,
                dec?.rvProgress,
                dec?.cardResults
            )
        )
    }

    fun clearFieldValue() {
        enc?.etAesKey?.clear()
        dec?.etAesKey?.clear()
    }
```

```
    clearPropertiesValue()
    clearFieldValue()
    hideOperation()
    initValueOfView()
}

/**
 * UPLOAD FILE STAGES
 * */

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    /**
     * Set [fis] property and [outputFileEncrypt] for encrypt operation
     * */
    fun setDataFileForEncrypt() {
        try {
            data?.data?.let { uri ->
                val path = UriUtil.getPathFromUri(requireContext(), uri)
                val file = path?.let { File(it) }
                enc?.tvUploadFile?.text = getString(R.string.choose_file, file?.path)
                originalSize = file?.sizeStrWithBytes()
                fis = context?.contentResolver?.openInputStream(uri)
                outputFileEncrypt = setOutputFile(
                    Operation.ENCRYPT,
                    file?.name!!,
                    prefixAesEncryptFileName,
                    prefixAesDecryptFileName
                )
            }
        } catch (e: Exception) {
            log(exception = e) // tracking log error
            context?.showToast(getString(R.string.error_upload))
            enc?.tvUploadFile?.text = getString(R.string.choose_file, questionMark)
        }
    }

    /**
     * Set [fis] property and [outputFileDecrypt] for decrypt operation
     * */
    fun setDataFileForDecrypt() {
        try {
            data?.data?.let { uri ->
                val path = UriUtil.getPathFromUri(requireContext(), uri)
                val file = path?.let { File(it) }
                dec?.tvUploadFile?.text = getString(R.string.choose_file, file?.path)
                originalSize = file?.sizeStrWithBytes()
                fis = context?.contentResolver?.openInputStream(uri)
                outputFileDecrypt = setOutputFile(
                    Operation.DECRYPT,
                    file?.name!!,
                    prefixAesEncryptFileName,
                    prefixAesDecryptFileName
                )
            }
        } catch (e: Exception) {
            log(exception = e) // tracking log error
            context?.showToast(getString(R.string.error_upload))
            dec?.tvUploadFile?.text = getString(R.string.choose_file, questionMark)
        }
    }
```

```
    if (resultCode != Activity.RESULT_OK) return
    when (requestCode) {
        requestCodeEncrypt -> setDataFileForEncrypt()
        requestCodeDecrypt -> setDataFileForDecrypt()
    }
}

/**
 * ENCRYPTION STAGES
 * */

private fun useRandomPassword() {
    val randomPassword = getRandomString()
    enc?.etAesKey?.setText(randomPassword)
}

private fun doEncrypt() {
    if (enc?.etLayoutAesKey.checkFieldIsEmpty()) return
    if (isContainQuestionMark(enc?.tvUploadFile?.text.toString())) {
        context?.showToast(getString(R.string.not_choose_file))
        return
    }

    clearExistingFile(outputFileEncrypt)
    val password = enc?.etAesKey?.text.toString()
    setVisibleStateOfView(true, arrayListOf(enc?.titleProgress, enc?.rvProgress))
    progressOperation.push(Operation.ENCRYPT, getString(R.string.prepare_document_encrypt))
    Log.w("BlackBox", "Time start")

    benchmarkTimerEncrypt.startLog()
    aesSymmetric.encrypt(fis, password, encryptFileResultListener, outputFileEncrypt)
    enc?.etAesKey?.clear()
}

private val encryptFileResultListener = object : ECResultListener {
    override fun onProgress(newBytes: Int, bytesProcessed: Long, totalBytes: Long) {
        Log.w("BlackBox", "Time calculation: encryptFileResultListener#onProgress")
        progressOperation.push(Operation.ENCRYPT, getString(R.string.waiting_encrypt_doc))
    }

    override fun onFailure(message: String, e: Exception) {
        log(exception = e) // tracking log error
        progressOperation.push(Operation.ENCRYPT, e.message.toString(), exception = e)
        activity?.runOnUiThread { showFailedEncrypt() }
    }

    override fun <T> onSuccess(result: T) {
        Log.w("BlackBox", "Time End: encryptFileResultListener#onSuccess")
        benchmarkTimerEncrypt.endLog()
        val info = getString(R.string.success_encrypt_doc)
        progressOperation.push(Operation.ENCRYPT, info, finish = true)
        activity?.runOnUiThread { showResultEncrypt() }
    }
}

private fun showResultEncrypt() {
    enc?.apply {
        val sizeResult = outputFileEncrypt.sizeStrWithBytes()
        val processTime = benchmarkTimerEncrypt.getBenchmarkTime()
```

```
            tvSizeDefault.text = getString(R.string.result_size_default, originalSize)
            tvSizeResult.text = getString(R.string.result_size_encrypt, sizeResult)
            tvProcessTime.text = getString(R.string.result_time_encrypt, processTime)
        }
        setColorStateOfCard(
            Process.SUCCESS,
            enc?.cardResults,
            enc?.containerCardResults,
            enc?.titleResult,
            enc?.tvSizeDefault,
            enc?.tvSizeResult,
            enc?.tvProcessTime
        )
        setVisibleStateOfView(true, arrayListOf(enc?.cardResults))
        showSnackBar(
            message = getString(R.string.snackbar_info_encrypt),
            actionText = getString(R.string.snackbar_action_close),
            duration = Snackbar.LENGTH_INDEFINITE
        )
    }

    private fun showFailedEncrypt() {
        enc?.apply {
            tvSizeDefault.text = getString(R.string.result_size_default, questionMark)
            tvSizeResult.text = getString(R.string.result_size_encrypt, questionMark)
            tvProcessTime.text = getString(R.string.result_time_encrypt, questionMark)
        }
        setColorStateOfCard(
            Process.FAILED,
            enc?.cardResults,
            enc?.containerCardResults,
            enc?.titleResult,
            enc?.tvSizeDefault,
            enc?.tvSizeResult,
            enc?.tvProcessTime
        )
        setVisibleStateOfView(true, arrayListOf(enc?.cardResults))
    }

    /**
     * DECRYPTION STAGES
     * */

    private fun doDecrypt() {
        if (dec?.etLayoutAesKey.checkFieldIsEmpty()) return
        if (isContainQuestionMark(dec?.tvUploadFile?.text.toString())) {
            context?.showToast(getString(R.string.not_choose_file))
            return
        }

        clearExistingFile(outputFileDecrypt)
        val password = dec?.etAesKey?.text.toString()
        setVisibleStateOfView(true, arrayListOf(dec?.titleProgress, dec?.rvProgress))
        progressOperation.push(Operation.DECRYPT, getString(R.string.prepare_document_decrypt))
        Log.w("BlackBox", "Time start")

        benchmarkTimerDecrypt.startLog()
        aesSymmetric.decrypt(fis, password, decryptFileResultListener, outputFileDecrypt)
        dec?.etAesKey?.clear()
    }
```

```
private val decryptFileResultListener = object : ECResultListener {
    override fun onProgress(newBytes: Int, bytesProcessed: Long, totalBytes: Long) {
        Log.w("BlackBox", "Time calculation: decryptFileResultListener#onProgress")
        progressOperation.push(Operation.DECRYPT, getString(R.string.waiting_decrypt_doc))
    }

    override fun onFailure(message: String, e: Exception) {
        log(exception = e) // tracking log error
        progressOperation.push(Operation.DECRYPT, e.message.toString(), exception = e)
        activity?.runOnUiThread { showFailedDecrypt() }
    }

    override fun <T> onSuccess(result: T) {
        Log.w("BlackBox", "Time End: decryptFileResultListener#onSuccess")
        benchmarkTimerDecrypt.endLog()
        val info = getString(R.string.success_decrypt_doc)
        progressOperation.push(Operation.DECRYPT, info, finish = true)
        activity?.runOnUiThread { showResultDecrypt() }
    }
}

private fun showResultDecrypt() {
    dec?.apply {
        val sizeResult = outputFileDecrypt.sizeStrWithBytes()
        val processTime = benchmarkTimerDecrypt.getBenchmarkTime()
        tvSizeDefault.text = getString(R.string.result_size_encrypt, originalSize)
        tvSizeResult.text = getString(R.string.result_size_decrypt, sizeResult)
        tvProcessTime.text = getString(R.string.result_time_decrypt, processTime)
    }
    setColorStateOfCard(
        Process.SUCCESS,
        dec?.cardResults,
        dec?.containerCardResults,
        dec?.titleResult,
        dec?.tvSizeDefault,
        dec?.tvSizeResult,
        dec?.tvProcessTime
    )
    setVisibleStateOfView(true, arrayListOf(dec?.cardResults))
    showSnackBar(
        message = getString(R.string.snackbar_info_decrypt),
        actionText = getString(R.string.snackbar_action_close),
        duration = Snackbar.LENGTH_INDEFINITE
    )
}

private fun showFailedDecrypt() {
    dec?.apply {
        tvSizeDefault.text = getString(R.string.result_size_encrypt, questionMark)
        tvSizeResult.text = getString(R.string.result_size_decrypt, questionMark)
        tvProcessTime.text = getString(R.string.result_time_decrypt, questionMark)
    }
    setColorStateOfCard(
        Process.FAILED,
        dec?.cardResults,
        dec?.containerCardResults,
        dec?.titleResult,
        dec?.tvSizeDefault,
        dec?.tvSizeResult,
```

```
            dec?.tvProcessTime
        )
        setVisibleStateOfView(true, arrayListOf(dec?.cardResults))
    }
}
```

Lampiran 3. Kode Program RSA

```
package com.yogaprasetyo.skripsi.keepsecret.ui.rsa

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.core.view.isVisible
import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.google.android.material.snackbar.Snackbar
import com.yogaprasetyo.skripsi.keepsecret.R
import com.yogaprasetyo.skripsi.keepsecret.adapter.ProgressAdapter
import com.yogaprasetyo.skripsi.keepsecret.databinding.DecryptRsaOfflineBinding
import com.yogaprasetyo.skripsi.keepsecret.databinding.EncryptRsaOfflineBinding
import com.yogaprasetyo.skripsi.keepsecret.databinding.FragmentRsaOfflineBinding
import com.yogaprasetyo.skripsi.keepsecret.model.Progress
import com.yogaprasetyo.skripsi.keepsecret.utils.*
import java.io.File
import java.io.IOException
import java.io.InputStream
import java.security.KeyPair

class RsaOfflineFragment : Fragment() {

    private lateinit var progressOperation: ProgressOperation
    private lateinit var progressAdapterEncrypt: ProgressAdapter
    private lateinit var progressAdapterDecrypt: ProgressAdapter

    private lateinit var outputFileEncrypt: File
    private lateinit var outputFileDecrypt: File

    private lateinit var sharedPref: KeySharedPreferences
    private lateinit var sharedPublicKey: String

    private val questionMark by lazy { getString(R.string.question_mark) }
    private val rsaKeys by lazy { ECKeys() }
    private val rsaAsymmetric by lazy { ECAsymmetric() }
    private val progressArray by lazy { ArrayList<Progress>() }

    private val benchmarkTimerEncrypt by lazy {
        BenchmarkTimer("Rsa", "Encrypt operation")
    }
    private val benchmarkTimerDecrypt by lazy {
        BenchmarkTimer("Rsa", "Decrypt operation")
    }
```

```
private var fis: InputStream? = null
private var originalSize: String? = null

private var enc: EncryptRsaOfflineBinding? = null
private var dec: DecryptRsaOfflineBinding? = null

private var _binding: FragmentRsaOfflineBinding? = null
private val binding get() = _binding

override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    _binding = FragmentRsaOfflineBinding.inflate(inflater, container, false)
    return binding?.root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    initProperties()
    initRecyclerView()
    initButtonListener()
    initValueOfView()
    initCheckedChangeListener(Operation.ENCRYPT)
    initCheckedChangeListener(Operation.DECRYPT)
}

private fun initProperties() {
    enc = binding?.eoperation
    dec = binding?.doperation
    sharedPref = KeySharedPreferences(requireContext())
    progressAdapterEncrypt = ProgressAdapter()
    progressAdapterDecrypt = ProgressAdapter()
    progressOperation = ProgressOperation(
        progressArray,
        progressAdapterEncrypt,
        progressAdapterDecrypt
    )
}

private fun initCheckedChangeListener(operation: Operation) {
    fun setListenerEncrypt() {
        enc?.cbQuestion?.checkedChangeListener(
            arrayListOf(enc?.cardInfo, enc?.etLayoutPublicKey),
            arrayListOf(
                enc?.titleGenerateRsaKey,
                enc?.cardGenerate,
                enc?.tvStatusSaveKey,
                enc?.btnGenerate
            )
        )
    }

    fun setListenerDecrypt() {
        dec?.cbQuestion?.checkedChangeListener(
            arrayListOf(
                dec?.cardInfo,
                dec?.titleGenerateRsaKey,
```

```
                dec?.cardGenerate,
                dec?.btnGenerate,
                dec?.btnCopy,
            )
        )
    }

    when (operation) {
        Operation.ENCRYPT -> setListenerEncrypt()
        Operation.DECRYPT -> setListenerDecrypt()
    }
}

private fun initRecyclerView() {
    fun configure(rv: RecyclerView): RecyclerView {
        return rv.apply {
            layoutManager = LinearLayoutManager(activity)
            setHasFixedSize(true)
        }
    }

    enc?.rvProgress?.let { configure(it).adapter = progressAdapterEncrypt }
    dec?.rvProgress?.let { configure(it).adapter = progressAdapterDecrypt }
}

private fun initButtonListener() {
    /**
     * Showing operation
     * */
    binding?.btnToggleShowOperation?.toggleButtonListener(
        {
            showCryptoOperation(Operation.ENCRYPT)
            resetAllCacheOrValue()
        },
        {
            showCryptoOperation(Operation.DECRYPT)
            resetAllCacheOrValue()
        }
    )

    /**
     * Encrypt Operation
     * */
    enc?.btnGenerate?.setOnClickListener { generateLocalKeyPair() }
    enc?.btnUpload?.setOnClickListener { uploadFile(Operation.ENCRYPT) }
    enc?.btnEncrypt?.setOnClickListener { doEncrypt() }

    /**
     * Decrypt Operation
     * */
    dec?.btnGenerate?.setOnClickListener { generateSharedKeyPair() }
    dec?.btnCopy?.setOnClickListener { copySharedPubKey() }
    dec?.btnUpload?.setOnClickListener { uploadFile(Operation.DECRYPT) }
    dec?.btnDecrypt?.setOnClickListener { doDecrypt() }
}

private fun showCryptoOperation(operationName: Operation) {
    fun showEncrypt(isVisible: Boolean = true) {
        binding?.eoperation?.root?.isVisible = isVisible
    }
```

```
    fun showDecrypt(isVisible: Boolean = true) {
        binding?.doperation?.root?.isVisible = isVisible
    }

    when (operationName) {
        Operation.ENCRYPT -> {
            showEncrypt()
            showDecrypt(false)
        }
        Operation.DECRYPT -> {
            showDecrypt()
            showEncrypt(false)
        }
    }
}

private fun initValueOfView() {
    val privateKey = getString(R.string.private_key, questionMark)
    val publicKey = getString(R.string.public_key, questionMark)
    val chosenFile = getString(R.string.choose_file, questionMark)
    val processGenerateKey = getString(R.string.state_generated, getString(R.string.check))

    enc?.apply {
        tvPrivateKey.text = privateKey
        tvPublicKey.text = publicKey
        tvStatusSaveKey.text = processGenerateKey
        tvUploadFile.text = chosenFile
    }

    dec?.apply {
        tvStatusSaveKey.text = processGenerateKey
        tvPublicKey.text = publicKey
        tvUploadFile.text = chosenFile
    }
}

private fun resetAllCacheOrValue() {
    fun clearPropertiesValue() {
        fis = null
        originalSize = null
        progressArray.clear()
    }

    fun hideOperation() {
        setVisibleStateOfView(
            false,
            arrayListOf(
                enc?.titleProgress,
                enc?.rvProgress,
                enc?.cardResults,
                dec?.titleProgress,
                dec?.rvProgress,

                dec?.cardResults
            )
        )
    }

    fun clearFieldValue() {
```

```
        enc?.etGenerateKey?.clear()
    }

    clearPropertiesValue()
    clearFieldValue()
    hideOperation()
    initValueOfView()
}

/**
 * UPLOAD FILE STAGES
 * */

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    /**
     * Set [fis] property and [outputFileEncrypt] for encrypt operation
     * */
    fun setDataFileForEncrypt() {
        try {
            data?.data?.let { uri ->
                val path = UriUtil.getPathFromUri(requireContext(), uri)
                val file = path?.let { File(it) }
                enc?.tvUploadFile?.text = getString(R.string.choose_file, file?.path)
                originalSize = file?.sizeStrWithBytes()
                fis = context?.contentResolver?.openInputStream(uri)
                outputFileEncrypt = setOutputFile(
                    Operation.ENCRYPT,
                    file?.name!!,
                    prefixRsaEncryptFileName,
                    prefixRsaDecryptFileName
                )
            }
        } catch (e: Exception) {
            log(exception = e) // tracking log error
            context?.showToast(getString(R.string.error_upload))
            enc?.tvUploadFile?.text = getString(R.string.choose_file, questionMark)
        }
    }

    /**
     * Set [fis] property and [outputFileDecrypt] for decrypt operation
     * */
    fun setDataFileForDecrypt() {
        try {
            data?.data?.let { uri ->
                val path = UriUtil.getPathFromUri(requireContext(), uri)
                val file = path?.let { File(it) }
                dec?.tvUploadFile?.text = getString(R.string.choose_file, file?.path)
                originalSize = file?.sizeStrWithBytes()
                fis = context?.contentResolver?.openInputStream(uri)
                outputFileDecrypt = setOutputFile(
                    Operation.DECRYPT,
                    file?.name!!,
                    prefixRsaEncryptFileName,
                    prefixRsaDecryptFileName
                )
            }
        } catch (e: Exception) {
            log(exception = e) // tracking log error
            context?.showToast(getString(R.string.error_upload))
```

```
            dec?.tvUploadFile?.text = getString(R.string.choose_file, questionMark)
        }
    }

    if (resultCode != Activity.RESULT_OK) return
    when (requestCode) {
        requestCodeEncrypt -> setDataFileForEncrypt()
        requestCodeDecrypt -> setDataFileForDecrypt()
    }
}

/**
 * ENCRYPTION STAGES
 * */

private fun generateLocalKeyPair() {
    fun setKeyPairViewText() {
        val pairRsaKey = sharedPref.getLocalKeyRsa()
        enc?.tvPrivateKey?.text = getString(R.string.private_key, pairRsaKey.first)
        enc?.tvPublicKey?.text = getString(R.string.public_key, pairRsaKey.second)
        setTextStateGenerateKey(Process.INFO)
    }

    when {
        sharedPref.isKeyRsaOfLocalAvailable() -> setKeyPairViewText()
        else -> rsaKeys.genRSAKeyPair(rsaEncryptKeyPairListener, ECAsymmetric.KeySizes.S_2048)
    }
}

private val rsaEncryptKeyPairListener = object : ECRSAKeyPairListener {
    override fun onFailure(message: String, e: Exception) {
        log(exception = e) // tracking log error
        activity?.runOnUiThread {
            setColorStateOfCard(
                Process.FAILED,
                enc?.cardGenerate,
                enc?.containerCardGenerate,
                enc?.tvPrivateKey,
                enc?.tvPublicKey
            )
            setTextStateGenerateKey(Process.FAILED)
        }
    }

    override fun onGenerated(keyPair: KeyPair) {
        val pubKey = keyPair.public.encoded.toBase64String()
        val privKey = keyPair.private.encoded.toBase64String()
        sharedPref.saveLocalKeyRsa(pubKey, privKey)
        activity?.runOnUiThread {
            enc?.tvPrivateKey?.text = getString(R.string.private_key, privKey)
            enc?.tvPublicKey?.text = getString(R.string.public_key, pubKey)
            setTextStateGenerateKey(Process.SUCCESS)
        }
    }
}

private fun setTextStateGenerateKey(process: Process) {
    enc?.tvStatusSaveKey?.text = when (process) {
        Process.SUCCESS -> getString(R.string.state_generated, getString(R.string.success))
        Process.FAILED -> getString(R.string.state_generated, getString(R.string.failed))
```

```
            else -> getString(R.string.is_key_already)
        }
    }

    private fun doEncrypt() {
        val isFileShared = enc?.cbQuestion?.isChecked == true
        if (isFileShared) {
            if (enc?.etLayoutPublicKey.checkFieldIsEmpty()) return
            if (isContainQuestionMark(enc?.tvUploadFile?.text.toString())) {
                context?.showToast(getString(R.string.not_choose_file))
                return
            }
        } else {
            if (
                isContainQuestionMark(enc?.tvUploadFile?.text.toString())
                || (isContainQuestionMark(enc?.tvPrivateKey?.text.toString())
                        && isContainQuestionMark(enc?.tvPublicKey?.text.toString()))
            ) {
                context?.showToast(getString(R.string.empty_key_and_file))
                return
            }
        }

        clearExistingFile(outputFileEncrypt)
        setVisibleStateOfView(true, arrayListOf(enc?.titleProgress, enc?.rvProgress))
        progressOperation.push(Operation.ENCRYPT, getString(R.string.prepare_document_encrypt))

        try {
            val pubKey = when (enc?.cbQuestion?.isChecked) {
                true -> rsaKeys.genRSAPublicKeyFromBase64(enc?.etGenerateKey?.text.toString())
                else -> rsaKeys.genRSAPublicKeyFromBase64(sharedPref.getLocalKeyRsa().second!!)
            }

            Log.w("BlackBox", "Time Start")
            benchmarkTimerEncrypt.startLog()
            rsaAsymmetric.encrypt(fis, pubKey, encryptFileResultListener, outputFileEncrypt)
        } catch (e: Exception) {
            log(exception = e)
            progressOperation.push(
                Operation.ENCRYPT,
                getString(R.string.error_operation, e.message.toString()),
                exception = e
            )
            showFailedEncrypt()
        }
    }

    private val encryptFileResultListener = object : ECResultListener {
        override fun onProgress(newBytes: Int, bytesProcessed: Long, totalBytes: Long) {
            Log.w("BlackBox", "Time Calculation: encryptFileResultListener#onProgress")
            progressOperation.push(Operation.ENCRYPT, getString(R.string.waiting_encrypt_doc))
        }

        override fun onFailure(message: String, e: Exception) {
            log(exception = e) // tracking log error
            progressOperation.push(Operation.ENCRYPT, e.message.toString(), exception = e)
            activity?.runOnUiThread { showFailedEncrypt() }
        }

        override fun <T> onSuccess(result: T) {
```

```
            Log.w("BlackBox", "Time End: encryptFileResultListener#onSuccess")
            benchmarkTimerEncrypt.endLog()
            val info = getString(R.string.success_encrypt_doc)
            progressOperation.push(Operation.ENCRYPT, info, finish = true)
            activity?.runOnUiThread { showResultEncrypt() }
        }
    }

    private fun showResultEncrypt() {
        enc?.apply {
            val sizeResult = outputFileEncrypt.sizeStrWithBytes()
            val processTime = benchmarkTimerEncrypt.getBenchmarkTime()
            tvSizeDefault.text = getString(R.string.result_size_default, originalSize)
            tvSizeResult.text = getString(R.string.result_size_encrypt, sizeResult)
            tvProcessTime.text = getString(R.string.result_time_encrypt, processTime)
        }
        setColorStateOfCard(
            Process.SUCCESS,
            enc?.cardResults,
            enc?.containerCardResults,
            enc?.titleResult,
            enc?.tvSizeDefault,
            enc?.tvSizeResult,
            enc?.tvProcessTime
        )
        setVisibleStateOfView(true, arrayListOf(enc?.cardResults))
        showSnackBar(
            message = getString(R.string.snackbar_info_encrypt),
            actionText = getString(R.string.snackbar_action_close),
            duration = Snackbar.LENGTH_INDEFINITE
        )
    }

    private fun showFailedEncrypt() {
        enc?.apply {
            tvSizeDefault.text = getString(R.string.result_size_default, questionMark)
            tvSizeResult.text = getString(R.string.result_size_encrypt, questionMark)
            tvProcessTime.text = getString(R.string.result_time_encrypt, questionMark)
        }
        setColorStateOfCard(
            Process.FAILED,
            enc?.cardResults,
            enc?.containerCardResults,
            enc?.titleResult,
            enc?.tvSizeDefault,
            enc?.tvSizeResult,
            enc?.tvProcessTime
        )
        setVisibleStateOfView(true, arrayListOf(enc?.cardResults))
    }

    /**
     * DECRYPTION STAGES
     * */

    private fun generateSharedKeyPair() {
        fun setKeyPairViewText() {
            val pairRsaKey = sharedPref.getSharedKeyRsa()
            sharedPublicKey = pairRsaKey.second!!
            dec?.btnCopy?.isEnabled = true
```

```
            dec?.tvPublicKey?.text = pairRsaKey.second
            dec?.tvStatusSaveKey?.text = getString(R.string.is_key_already)
        }

        when {
            sharedPref.isKeyRsaOfSharedAvailable() -> setKeyPairViewText()
            else -> rsaKeys.genRSAKeyPair(rsaDecryptKeyPairListener, ECAsymmetric.KeySizes.S_2048)
        }
    }

    private val rsaDecryptKeyPairListener = object : ECRSAKeyPairListener {
        override fun onFailure(message: String, e: Exception) {
            log(exception = e) // tracking log error
            activity?.runOnUiThread {
                setColorStateOfCard(
                    Process.FAILED,
                    dec?.cardGenerate,
                    dec?.containerCardGenerate,
                    dec?.tvStatusSaveKey,
                    dec?.tvPublicKey
                )
                dec?.tvStatusSaveKey?.text =
                    getString(R.string.state_generated, getString(R.string.failed))
            }
        }

        override fun onGenerated(keyPair: KeyPair) {
            val publicKey = keyPair.public.encoded.toBase64String()
            val privateKey = keyPair.private.encoded.toBase64String()
            sharedPref.saveSharedKeyRsa(publicKey, privateKey)
            sharedPublicKey = publicKey
            activity?.runOnUiThread {
                dec?.tvPublicKey?.text = publicKey
                dec?.btnCopy?.isEnabled = true
                dec?.tvStatusSaveKey?.text =
                    getString(R.string.state_generated, getString(R.string.success))
            }
        }
    }

    private fun copySharedPubKey() {
        if (!this::sharedPublicKey.isInitialized) return
        context?.copyToClipboard(sharedPublicKey)
    }

    private fun doDecrypt() {
        val isFileFromOthers = dec?.cbQuestion?.isChecked == true
        if (isFileFromOthers) {
            if (
                isContainQuestionMark(dec?.tvPublicKey?.text.toString())
                || isContainQuestionMark(dec?.tvUploadFile?.text.toString())
            ) {
                context?.showToast(getString(R.string.empty_key_and_file))
                return
            }
        } else {
            if (isContainQuestionMark(dec?.tvUploadFile?.text.toString())) return
        }

        val sharedPrefPrivateKey = when {
```

```
            isFileFromOthers -> sharedPref.getSharedKeyRsa()
            else -> sharedPref.getLocalKeyRsa()
        }

        setVisibleStateOfView(true, arrayListOf(dec?.titleProgress, dec?.rvProgress))
        try {
            val privateKey = rsaKeys.genRSAPrivateKeyFromBase64(sharedPrefPrivateKey.first!!)
            progressOperation.push(Operation.DECRYPT, getString(R.string.prepare_document_decrypt))
            Log.w("BlackBox", "Time Start")
            benchmarkTimerDecrypt.startLog()
            rsaAsymmetric.decrypt(fis, privateKey, decryptFileResultListener)
        } catch (e: Exception) {
            log(exception = e) // tracking log error
            progressOperation.push(Operation.DECRYPT, e.message.toString(), exception = e)
            showFailedDecrypt()
        }
    }

    private val decryptFileResultListener = object : ECResultListener {
        override fun onProgress(newBytes: Int, bytesProcessed: Long, totalBytes: Long) {
            Log.w("BlackBox", "Time Calculation: decryptFileResultListener#onProgress")
            progressOperation.push(Operation.ENCRYPT, getString(R.string.waiting_decrypt_doc))
        }

        override fun onFailure(message: String, e: Exception) {
            log(exception = e) // tracking log error
            progressOperation.push(Operation.DECRYPT, e.message.toString(), exception = e)
            activity?.runOnUiThread { showFailedDecrypt() }
        }

        override fun <T> onSuccess(result: T) {
            Log.w("BlackBox", "Time End: decryptFileResultListener#onSuccess")
            benchmarkTimerDecrypt.endLog()
            val info = getString(R.string.success_decrypt_doc)
            progressOperation.push(Operation.DECRYPT, info, finish = true)

            try {
                /**
                 * Because we can't use a custom save directory file, we renaming
                 * the result file from decrypt operation to a target directory. In order that
                 * file result still can grouping in one folder.
                 *
                 * Example path
                 * ``` root/Documents/Decrypt Files/Dec RSA_filename.extension```
                 * */
                val file = result as File
                val isRenamed = file.renameTo(outputFileDecrypt)
                if (isRenamed) {
                    log("Rename To Success")
                } else {
                    log("Rename To Failed")
                }

                activity?.runOnUiThread { showResultDecrypt() }
            } catch (ioException: IOException) {
                log(exception = ioException)
            } catch (e: Exception) {
                log(exception = e)
            }
        }
```

```
    }

    private fun showResultDecrypt() {
        dec?.apply {
            val sizeResult = outputFileDecrypt.sizeStrWithBytes()
            val processTime = benchmarkTimerDecrypt.getBenchmarkTime()
            tvSizeDefault.text = getString(R.string.result_size_encrypt, originalSize)
            tvSizeResult.text = getString(R.string.result_size_decrypt, sizeResult)
            tvProcessTime.text = getString(R.string.result_time_decrypt, processTime)
        }
        setColorStateOfCard(
            Process.SUCCESS,
            dec?.cardResults,
            dec?.containerCardResults,
            dec?.titleResult,
            dec?.tvSizeDefault,
            dec?.tvSizeResult,
            dec?.tvProcessTime
        )
        setVisibleStateOfView(true, arrayListOf(dec?.cardResults))
        showSnackBar(
            message = getString(R.string.snackbar_info_decrypt),
            actionText = getString(R.string.snackbar_action_close),
            duration = Snackbar.LENGTH_INDEFINITE
        )
    }

    private fun showFailedDecrypt() {
        dec?.apply {
            tvSizeDefault.text = getString(R.string.result_size_encrypt, questionMark)
            tvSizeResult.text = getString(R.string.result_size_decrypt, questionMark)
            tvProcessTime.text = getString(R.string.result_time_decrypt, questionMark)
        }
        setColorStateOfCard(
            Process.FAILED,
            dec?.cardResults,
            dec?.containerCardResults,
            dec?.titleResult,
            dec?.tvSizeDefault,
            dec?.tvSizeResult,
            dec?.tvProcessTime
        )
        setVisibleStateOfView(true, arrayListOf(dec?.cardResults))
    }
}
```

Lampiran 4. Kode Program Kombinasi

```
package com.yogaprasetyo.skripsi.keepsecret.ui.hybrid

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.core.view.isVisible
```

```
import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.google.android.material.snackbar.Snackbar
import com.yogaprasetyo.skripsi.keepsecret.R
import com.yogaprasetyo.skripsi.keepsecret.adapter.ProgressAdapter
import com.yogaprasetyo.skripsi.keepsecret.databinding.DecryptHybridOfflineBinding
import com.yogaprasetyo.skripsi.keepsecret.databinding.EncryptHybridOfflineBinding
import com.yogaprasetyo.skripsi.keepsecret.databinding.FragmentHybridOfflineBinding
import com.yogaprasetyo.skripsi.keepsecret.model.Progress
import com.yogaprasetyo.skripsi.keepsecret.utils.*
import com.yogaprasetyo.skripsi.keepsecret.utils.UriUtil.getPathFromUri
import java.io.File
import java.io.InputStream
import java.security.KeyPair
import java.security.interfaces.RSAPublicKey

class HybridOfflineFragment : Fragment() {

    private lateinit var password: String
    private lateinit var passwordEncrypted: String
    private lateinit var publicKey: RSAPublicKey
    private lateinit var sharedPublicKey: String

    private lateinit var outputFileEncrypt: File
    private lateinit var outputFileDecrypt: File

    private lateinit var sharedPref: KeySharedPreferences
    private lateinit var originalFile: String // alias key for local password encrypted

    private lateinit var progressOperation: ProgressOperation
    private lateinit var progressAdapterEncrypt: ProgressAdapter
    private lateinit var progressAdapterDecrypt: ProgressAdapter

    private val questionMark by lazy { getString(R.string.question_mark) }
    private val rsaKeys by lazy { ECKeys() }
    private val aesSymmetric by lazy { ECSymmetric() }
    private val rsaAsymmetric by lazy { ECAsymmetric() }
    private val progressArray by lazy { ArrayList<Progress>() }

    private val benchmarkTimerEncrypt by lazy {
        BenchmarkTimer("Hybrid", "Encrypt operation")
    }
    private val benchmarkTimerDecrypt by lazy {
        BenchmarkTimer("Hybrid", "Decrypt operation")
    }

    private var fis: InputStream? = null
    private var originalSize: String? = null

    private var enc: EncryptHybridOfflineBinding? = null
    private var dec: DecryptHybridOfflineBinding? = null

    private var _binding: FragmentHybridOfflineBinding? = null
    private val binding get() = _binding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
```

```
): View? {
    _binding = FragmentHybridOfflineBinding.inflate(inflater, container, false)
    return binding?.root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    initProperties()
    initRecyclerView()
    initButtonListener()
    initValueOfView()
    initCheckedChangeListener(Operation.ENCRYPT)
    initCheckedChangeListener(Operation.DECRYPT)
}

private fun initProperties() {
    enc = binding?.eoperation
    dec = binding?.doperation
    sharedPref = KeySharedPreferences(requireContext())
    progressAdapterEncrypt = ProgressAdapter()
    progressAdapterDecrypt = ProgressAdapter()
    progressOperation = ProgressOperation(
        progressArray,
        progressAdapterEncrypt,
        progressAdapterDecrypt
    )
}

private fun initCheckedChangeListener(operation: Operation) {
    fun setListenerEncrypt() {
        enc?.cbQuestion?.checkedChangeListener(
            arrayListOf(enc?.cardInfo, enc?.etLayoutPublicKey),
            arrayListOf(
                enc?.titleGenerateRsaKey,
                enc?.cardGenerate,
                enc?.tvStatusSaveKey,
                enc?.btnGenerate
            )
        )
    }

    fun setListenerDecrypt() {
        dec?.cbQuestion?.checkedChangeListener(
            arrayListOf(
                dec?.cardInfo,
                dec?.titleGenerateRsaKey,
                dec?.cardGenerate,
                dec?.btnGenerate,
                dec?.btnCopy,
                dec?.etLayoutAesKeyEncrypted
            )
        )
    }

    when (operation) {
        Operation.ENCRYPT -> setListenerEncrypt()
        Operation.DECRYPT -> setListenerDecrypt()
    }
}
```

```
private fun initRecyclerView() {
    fun configure(rv: RecyclerView): RecyclerView {
        return rv.apply {
            layoutManager = LinearLayoutManager(activity)
            setHasFixedSize(true)
        }
    }

    enc?.rvProgress?.let { configure(it).adapter = progressAdapterEncrypt }
    dec?.rvProgress?.let { configure(it).adapter = progressAdapterDecrypt }
}

private fun initButtonListener() {
    /**
     * Showing operation
     * */
    binding?.btnToggleShowOperation?.toggleButtonListener(
        {
            showCryptoOperation(Operation.ENCRYPT)
            resetAllCacheOrValue()
        },
        {
            showCryptoOperation(Operation.DECRYPT)
            resetAllCacheOrValue()
        }
    )

    /**
     * Encrypt Operation
     * */
    enc?.btnGenerate?.setOnClickListener { generateLocalKeyPair() }
    enc?.btnRandomKey?.setOnClickListener { useRandomPassword() }
    enc?.btnUpload?.setOnClickListener { uploadFile(Operation.ENCRYPT) }
    enc?.btnEncrypt?.setOnClickListener { doEncrypt() }
    enc?.btnCopy?.setOnClickListener { copyPasswordEncrypted() }

    /**
     * Decrypt Operation
     * */
    dec?.btnGenerate?.setOnClickListener { generateSharedKeyPair() }
    dec?.btnCopy?.setOnClickListener { copySharedPubKey() }
    dec?.btnUpload?.setOnClickListener { uploadFile(Operation.DECRYPT) }
    dec?.btnDecrypt?.setOnClickListener { doDecrypt() }
}

private fun showCryptoOperation(operationName: Operation) {
    fun showEncrypt(isVisible: Boolean = true) {
        binding?.eoperation?.root?.isVisible = isVisible
    }

    fun showDecrypt(isVisible: Boolean = true) {
        binding?.doperation?.root?.isVisible = isVisible
    }

    when (operationName) {
        Operation.ENCRYPT -> {
            showEncrypt()
            showDecrypt(false)
        }
```

```
            Operation.DECRYPT -> {
                showDecrypt()
                showEncrypt(false)
            }
        }
    }

    private fun initValueOfView() {
        val privateKey = getString(R.string.private_key, questionMark)
        val publicKey = getString(R.string.public_key, questionMark)
        val chosenFile = getString(R.string.choose_file, questionMark)
        val processGenerateKey = getString(R.string.state_generated, getString(R.string.check))

        enc?.apply {
            tvPrivateKey.text = privateKey
            tvPublicKey.text = publicKey
            tvStatusSaveKey.text = processGenerateKey
            tvUploadFile.text = chosenFile
        }

        dec?.apply {
            tvStatusSaveKey.text = processGenerateKey
            tvPublicKey.text = publicKey
            tvUploadFile.text = chosenFile
        }
    }

    private fun resetAllCacheOrValue() {
        fun clearPropertiesValue() {
            fis = null
            originalSize = null
            progressArray.clear()
        }

        fun hideOperation() {
            setVisibleStateOfView(
                false,
                arrayListOf(
                    enc?.titleProgress,
                    enc?.rvProgress,
                    enc?.titleEncryptedAesKey,
                    enc?.tvAesKeyEncrypted,
                    enc?.btnCopy,
                    enc?.cardResults,
                    dec?.titleProgress,
                    dec?.rvProgress,
                    dec?.cardResults
                )
            )
        }

        fun clearFieldValue() {
            enc?.etAesKey?.clear()
            enc?.etGenerateKey?.clear()
            dec?.etAesKeyEncrypted?.clear()
        }

        clearPropertiesValue()
        clearFieldValue()
        hideOperation()
```

```
    initValueOfView()
}

/**
 * UPLOAD FILE STAGES
 * */

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    /**
     * Set [fis] property and [outputFileEncrypt] for encrypt operation
     * */
    fun setDataFileForEncrypt() {
        try {
            data?.data?.let { uri ->
                val path = getPathFromUri(requireContext(), uri)
                val file = path?.let { File(it) }
                enc?.tvUploadFile?.text = getString(R.string.choose_file, file?.path)
                originalSize = file?.sizeStrWithBytes()
                fis = context?.contentResolver?.openInputStream(uri)
                outputFileEncrypt = setOutputFile(
                    Operation.ENCRYPT,
                    file?.name!!,
                    prefixHybridEncryptFileName,
                    prefixHybridDecryptFileName
                )
            }
        } catch (e: Exception) {
            log(exception = e) // tracking log error
            context?.showToast(getString(R.string.error_upload))
            enc?.tvUploadFile?.text = getString(R.string.choose_file, questionMark)
        }
    }

    /**
     * Set [fis] property and [outputFileDecrypt] for decrypt operation
     * */
    fun setDataFileForDecrypt() {
        try {
            data?.data?.let { uri ->
                val path = getPathFromUri(requireContext(), uri)
                val file = path?.let { File(it) }
                dec?.tvUploadFile?.text = getString(R.string.choose_file, file?.path)
                originalFile = file?.nameWithoutExtension.toString()
                originalSize = file?.sizeStrWithBytes()
                fis = context?.contentResolver?.openInputStream(uri)
                outputFileDecrypt = setOutputFile(
                    Operation.DECRYPT,
                    file?.name!!,
                    prefixHybridEncryptFileName,
                    prefixHybridDecryptFileName
                )
            }
        } catch (e: Exception) {
            log(exception = e) // tracking log error
            context?.showToast(getString(R.string.error_upload))
            dec?.tvUploadFile?.text = getString(R.string.choose_file, questionMark)
        }
    }

    if (resultCode != Activity.RESULT_OK) return
```

```
    when (requestCode) {
        requestCodeEncrypt -> setDataFileForEncrypt()
        requestCodeDecrypt -> setDataFileForDecrypt()
    }
}

/**
 * ENCRYPTION STAGES
 * */

private fun generateLocalKeyPair() {
    fun setKeyPairViewText() {
        val pairRsaKey = sharedPref.getLocalKeyHybrid()
        publicKey = rsaKeys.genRSAPublicKeyFromBase64(pairRsaKey.second!!)
        enc?.tvPrivateKey?.text = getString(R.string.private_key, pairRsaKey.first)
        enc?.tvPublicKey?.text = getString(R.string.public_key, pairRsaKey.second)
        setTextStateGenerateKey(Process.INFO)
    }

    when {
        sharedPref.isKeyHybridOfLocalAvailable() -> setKeyPairViewText()
        else -> rsaKeys.genRSAKeyPair(rsaEncryptKeyPairListener, ECAsymmetric.KeySizes.S_2048)
    }
}

private val rsaEncryptKeyPairListener = object : ECRSAKeyPairListener {
    override fun onFailure(message: String, e: Exception) {
        log(exception = e) // tracking log error
        setColorStateOfCard(
            Process.FAILED,
            enc?.cardGenerate,
            enc?.containerCardGenerate,
            enc?.tvPrivateKey,
            enc?.tvPublicKey
        )
        setTextStateGenerateKey(Process.FAILED)
    }

    override fun onGenerated(keyPair: KeyPair) {
        val pubKey = keyPair.public.encoded.toBase64String()
        val privKey = keyPair.private.encoded.toBase64String()
        sharedPref.saveLocalKeyHybrid(pubKey, privKey)
        activity?.runOnUiThread {
            publicKey = keyPair.public as RSAPublicKey
            enc?.tvPrivateKey?.text = getString(R.string.private_key, privKey)
            enc?.tvPublicKey?.text = getString(R.string.public_key, pubKey)
            setTextStateGenerateKey(Process.SUCCESS)
        }
    }
}

private fun setTextStateGenerateKey(process: Process) {
    enc?.tvStatusSaveKey?.text = when (process) {
        Process.SUCCESS -> getString(R.string.state_generated, getString(R.string.success))
        Process.FAILED -> getString(R.string.state_generated, getString(R.string.failed))
        else -> getString(R.string.is_key_already)
    }
}

private fun useRandomPassword() {
```

```
    val randomPassword = getRandomString()
    enc?.etAesKey?.setText(randomPassword)
}

private fun doEncrypt() {
    val isFileShared = enc?.cbQuestion?.isChecked == true
    if (isFileShared) {
        if (
            enc?.etLayoutPublicKey.checkFieldIsEmpty()
            || enc?.etLayoutAesKey.checkFieldIsEmpty()
        ) return

        if (isContainQuestionMark(enc?.tvUploadFile?.text.toString())) {
            context?.showToast(getString(R.string.not_choose_file))
            return
        }
    } else {
        if (enc?.etLayoutAesKey.checkFieldIsEmpty()) return
        if (
            isContainQuestionMark(enc?.tvUploadFile?.text.toString())
            || (isContainQuestionMark(enc?.tvPrivateKey?.text.toString())
                    && isContainQuestionMark(enc?.tvPublicKey?.text.toString()))
        ) {
            context?.showToast(getString(R.string.empty_key_and_file))
            return
        }
    }

    Log.i("BlackBox", "doEncrypt3: isFileShared=$isFileShared")
    clearExistingFile(outputFileEncrypt)
    password = enc?.etAesKey?.text.toString()
    setVisibleStateOfView(true, arrayListOf(enc?.titleProgress, enc?.rvProgress))
    progressOperation.push(Operation.ENCRYPT, getString(R.string.prepare_document_encrypt))

    Log.w("BlackBox", "Time Start")
    benchmarkTimerEncrypt.startLog()
    aesSymmetric.encrypt(fis, password, encryptFileResultListener, outputFileEncrypt)
}

private val encryptFileResultListener = object : ECResultListener {
    override fun onProgress(newBytes: Int, bytesProcessed: Long, totalBytes: Long) {
        Log.w("BlackBox", "Time Calculation: encryptFileResultListener#onProgress")
        progressOperation.push(Operation.ENCRYPT, getString(R.string.waiting_encrypt_doc))
    }

    override fun onFailure(message: String, e: Exception) {
        log(exception = e) // tracking log error
        progressOperation.push(Operation.ENCRYPT, e.message.toString(), exception = e)
        activity?.runOnUiThread { showFailedEncrypt() }
    }

    override fun <T> onSuccess(result: T) {
        Log.w("BlackBox", "Time Calculation: encryptFileResultListener#onSuccess")
        progressOperation.push(Operation.ENCRYPT, getString(R.string.success_encrypt_doc))
        progressOperation.push(Operation.ENCRYPT, getString(R.string.prepare_key_encrypt))

        try {
            val pubKey = when (enc?.cbQuestion?.isChecked) {
                true -> rsaKeys.genRSAPublicKeyFromBase64(enc?.etGenerateKey?.text.toString())
                else -> publicKey
```

```
            }
            rsaAsymmetric.encrypt(password, pubKey, encryptKeyResultListener)
        } catch (e: Exception) {
            val errorMessage = e.message.toString()
            Log.e("BlackBox", errorMessage)
            progressOperation.push(
                Operation.ENCRYPT,
                getString(R.string.error_operation, errorMessage),
                exception = e
            )
            activity?.runOnUiThread { showFailedEncrypt() }
            clearExistingFile(outputFileEncrypt)
        }
    }
}

private val encryptKeyResultListener = object : ECResultListener {
    override fun onProgress(newBytes: Int, bytesProcessed: Long, totalBytes: Long) {
        Log.w("BlackBox", "Time Calculation: encryptKeyResultListener#onProgress")
        progressOperation.push(Operation.ENCRYPT, getString(R.string.waiting_encrypt_key))
    }

    override fun onFailure(message: String, e: Exception) {
        log(exception = e) // tracking log error
        progressOperation.push(Operation.ENCRYPT, e.message.toString(), exception = e)
        activity?.runOnUiThread { showFailedEncrypt() }
    }

    override fun <T> onSuccess(result: T) {
        Log.w("BlackBox", "Time End: encryptKeyResultListener#onSuccess")
        benchmarkTimerEncrypt.endLog()
        val info = getString(R.string.success_encrypt_key)
        progressOperation.push(Operation.ENCRYPT, info, finish = true)

        passwordEncrypted = result as String
        val alias = outputFileEncrypt.nameWithoutExtension
        sharedPref.saveLocalAesKeyEncrypted(alias, passwordEncrypted)
        activity?.runOnUiThread { showResultEncrypt() }
    }
}

private fun showResultEncrypt() {
    enc?.apply {
        val sizeResult = outputFileEncrypt.sizeStrWithBytes()
        val processTime = benchmarkTimerEncrypt.getBenchmarkTime()
        tvSizeDefault.text = getString(R.string.result_size_default, originalSize)
        tvSizeResult.text = getString(R.string.result_size_encrypt, sizeResult)
        tvProcessTime.text = getString(R.string.result_time_encrypt, processTime)
        tvAesKeyEncrypted.text = passwordEncrypted
    }
    setColorStateOfCard(
        Process.SUCCESS,
        enc?.cardResults,
        enc?.containerCardResults,
        enc?.titleResult,
        enc?.tvSizeDefault,
        enc?.tvSizeResult,
        enc?.tvProcessTime
    )
    val arrayOfView = when (enc?.cbQuestion?.isChecked) {
```

```
            true -> arrayListOf(
                enc?.titleEncryptedAesKey,
                enc?.tvAesKeyEncrypted,
                enc?.btnCopy,
                enc?.cardResults
            )
            else -> {
                arrayListOf(enc?.cardResults)
            }
        }
        setVisibleStateOfView(true, arrayOfView)
        showSnackBar(
            message = getString(R.string.snackbar_info_encrypt),
            actionText = getString(R.string.snackbar_action_close),
            duration = Snackbar.LENGTH_INDEFINITE
        )
    }

    private fun showFailedEncrypt() {
        enc?.apply {
            tvSizeDefault.text = getString(R.string.result_size_default, questionMark)
            tvSizeResult.text = getString(R.string.result_size_encrypt, questionMark)
            tvProcessTime.text = getString(R.string.result_time_encrypt, questionMark)
            tvAesKeyEncrypted.text = questionMark
        }
        setColorStateOfCard(
            Process.FAILED,
            enc?.cardResults,
            enc?.containerCardResults,
            enc?.titleResult,
            enc?.tvSizeDefault,
            enc?.tvSizeResult,
            enc?.tvProcessTime
        )
        setVisibleStateOfView(true, arrayListOf(enc?.cardResults))
    }

    private fun copyPasswordEncrypted() {
        if (!this::passwordEncrypted.isInitialized) return
        context?.copyToClipboard(passwordEncrypted)
    }

    /**
     * DECRYPTION STAGES
     * */

    private fun generateSharedKeyPair() {
        fun setKeyPairViewText() {
            val pairRsaKey = sharedPref.getSharedKeyHybrid()
            sharedPublicKey = pairRsaKey.second!!
            dec?.btnCopy?.isEnabled = true
            dec?.tvPublicKey?.text = pairRsaKey.second
            dec?.tvStatusSaveKey?.text = getString(R.string.is_key_already)
        }

        when {
            sharedPref.isKeyHybridOfSharedAvailable() -> setKeyPairViewText()
            else -> rsaKeys.genRSAKeyPair(rsaDecryptKeyPairListener, ECAsymmetric.KeySizes.S_2048)
        }
    }
```

```
private val rsaDecryptKeyPairListener = object : ECRSAKeyPairListener {
    override fun onFailure(message: String, e: Exception) {
        log(exception = e) // tracking log error
        setColorStateOfCard(
            Process.FAILED,
            dec?.cardGenerate,
            dec?.containerCardGenerate,
            dec?.tvStatusSaveKey,
            dec?.tvPublicKey
        )
        dec?.tvStatusSaveKey?.text =
            getString(R.string.state_generated, getString(R.string.failed))
    }

    override fun onGenerated(keyPair: KeyPair) {
        val publicKey = keyPair.public.encoded.toBase64String()
        val privateKey = keyPair.private.encoded.toBase64String()
        sharedPref.saveSharedKeyHybrid(publicKey, privateKey)
        sharedPublicKey = publicKey
        activity?.runOnUiThread {
            dec?.tvPublicKey?.text = publicKey
            dec?.btnCopy?.isEnabled = true
            dec?.tvStatusSaveKey?.text =
                getString(R.string.state_generated, getString(R.string.success))
        }
    }
}

private fun copySharedPubKey() {
    if (!this::sharedPublicKey.isInitialized) return
    context?.copyToClipboard(sharedPublicKey)
}

private fun doDecrypt() {
    val isFileFromOthers = dec?.cbQuestion?.isChecked == true
    if (isFileFromOthers) {
        if (dec?.etLayoutAesKeyEncrypted.checkFieldIsEmpty()) return
        if (isContainQuestionMark(dec?.tvUploadFile?.text.toString())) {
            context?.showToast(getString(R.string.not_choose_file))
            return
        }
    } else {
        if (isContainQuestionMark(dec?.tvUploadFile?.text.toString())) {
            context?.showToast(getString(R.string.not_choose_file))
            return
        }
    }

    val sharedPrefPrivateKey = when {
        isFileFromOthers -> sharedPref.getSharedKeyHybrid()
        else -> sharedPref.getLocalKeyHybrid()
    }

    val passwordEncrypted = when {
        isFileFromOthers -> dec?.etAesKeyEncrypted?.text.toString()
        else -> sharedPref.getLocalAesKeyEncrypted(originalFile)
    }

    setVisibleStateOfView(true, arrayListOf(dec?.titleProgress, dec?.rvProgress))
```

```
    try {
        val privateKey = rsaKeys.genRSAPrivateKeyFromBase64(sharedPrefPrivateKey.first!!)
        progressOperation.push(Operation.DECRYPT, getString(R.string.prepare_key_decrypt))
        clearExistingFile(outputFileDecrypt)
        Log.w("BlackBox", "Time Start")
        benchmarkTimerDecrypt.startLog()
        rsaAsymmetric.decrypt(passwordEncrypted, privateKey, decryptKeyResultListener)
    } catch (e: Exception) {
        log(exception = e) // tracking log error
        progressOperation.push(Operation.DECRYPT, e.message.toString(), exception = e)
    }
}

private val decryptKeyResultListener = object : ECResultListener {
    override fun onProgress(newBytes: Int, bytesProcessed: Long, totalBytes: Long) {
        Log.w("BlackBox", "Time Calculation: decryptKeyResultListener#onProgress")
        progressOperation.push(Operation.DECRYPT, getString(R.string.waiting_decrypt_key))
    }

    override fun onFailure(message: String, e: Exception) {
        log(exception = e) // tracking log error
        progressOperation.push(Operation.DECRYPT, e.message.toString(), exception = e)
        activity?.runOnUiThread { showFailedDecrypt() }
    }

    override fun <T> onSuccess(result: T) {
        Log.w("BlackBox", "Time Calculation: decryptKeyResultListener#onSuccess")
        progressOperation.push(Operation.DECRYPT, getString(R.string.success_decrypt_key))
        progressOperation.push(Operation.DECRYPT, getString(R.string.prepare_document_decrypt))
        val password = result as String
        aesSymmetric.decrypt(fis, password, decryptFileResultListener, outputFileDecrypt)
    }
}

private val decryptFileResultListener = object : ECResultListener {
    override fun onProgress(newBytes: Int, bytesProcessed: Long, totalBytes: Long) {
        Log.w("BlackBox", "Time Calculation: decryptFileResultListener#onProgress")
        progressOperation.push(Operation.DECRYPT, getString(R.string.waiting_decrypt_doc))
    }

    override fun onFailure(message: String, e: Exception) {
        log(exception = e) // tracking error log
        progressOperation.push(Operation.DECRYPT, e.message.toString(), exception = e)
        activity?.runOnUiThread { showFailedDecrypt() }
    }

    override fun <T> onSuccess(result: T) {
        Log.w("BlackBox", "Time End: decryptFileResultListener#onSuccess")
        benchmarkTimerDecrypt.endLog()
        val info = getString(R.string.success_decrypt_doc)
        progressOperation.push(Operation.DECRYPT, info, finish = true)
        activity?.runOnUiThread { showResultDecrypt() }
    }
}

private fun showResultDecrypt() {
    dec?.apply {
        val sizeResult = outputFileDecrypt.sizeStrWithBytes()
        val processTime = benchmarkTimerDecrypt.getBenchmarkTime()
        tvSizeDefault.text = getString(R.string.result_size_encrypt, originalSize)
```

```
            tvSizeResult.text = getString(R.string.result_size_decrypt, sizeResult)
            tvProcessTime.text = getString(R.string.result_time_decrypt, processTime)
        }
        setColorStateOfCard(
            Process.SUCCESS,
            dec?.cardResults,
            dec?.containerCardResults,
            dec?.titleResult,
            dec?.tvSizeDefault,
            dec?.tvSizeResult,
            dec?.tvProcessTime
        )
        setVisibleStateOfView(
            true,
            arrayListOf(
                dec?.titleResult,
                dec?.cardResults
            )
        )
        showSnackBar(
            message = getString(R.string.snackbar_info_decrypt),
            actionText = getString(R.string.snackbar_action_close),
            duration = Snackbar.LENGTH_INDEFINITE
        )
    }

    private fun showFailedDecrypt() {
        dec?.apply {
            tvSizeDefault.text = getString(R.string.result_size_encrypt, questionMark)
            tvSizeResult.text = getString(R.string.result_size_decrypt, questionMark)
            tvProcessTime.text = getString(R.string.result_time_decrypt, questionMark)
        }
        setColorStateOfCard(
            Process.FAILED,
            dec?.cardResults,
            dec?.containerCardResults,
            dec?.titleResult,
            dec?.tvSizeDefault,
            dec?.tvSizeResult,
            dec?.tvProcessTime
        )
        setVisibleStateOfView(
            true,
            arrayListOf(
                dec?.titleResult,
                dec?.cardResults
            )
        )
    }
}
```

Skripsi Ganjil 22/23

ORIGINALITY REPORT

13%	13%	4%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	123dok.com Internet Source	1%
2	repositori.uin-alauddin.ac.id Internet Source	1%
3	etheses.uin-malang.ac.id Internet Source	1%
4	www.nrjetix.com Internet Source	1%
5	docplayer.info Internet Source	1%
6	Submitted to UC, Boulder Student Paper	<1%
7	repository.uinjkt.ac.id Internet Source	<1%
8	Submitted to South Dakota Board of Regents Student Paper	<1%
9	repository.its.ac.id Internet Source	<1%

10	people.etf.unsa.ba Internet Source	<1 %
11	ejurnal.seminar-id.com Internet Source	<1 %
12	eprints.utm.my Internet Source	<1 %
13	text-id.123dok.com Internet Source	<1 %
14	ejournal.ust.ac.id Internet Source	<1 %
15	jurnal.uisu.ac.id Internet Source	<1 %
16	repository.unej.ac.id Internet Source	<1 %
17	vddb.laba.lt Internet Source	<1 %
18	ijerat.com Internet Source	<1 %
19	www.ijser.org Internet Source	<1 %
20	media.neliti.com Internet Source	<1 %
21	doaj.org Internet Source	<1 %

22	documents.mx Internet Source	<1 %
23	nemertes.lis.upatras.gr Internet Source	<1 %
24	repositori.usu.ac.id Internet Source	<1 %
25	cs.uno.edu Internet Source	<1 %
26	eprints.akakom.ac.id Internet Source	<1 %
27	publications.lib.chalmers.se Internet Source	<1 %
28	citeseerx.ist.psu.edu Internet Source	<1 %
29	e-journal.potensi-utama.ac.id Internet Source	<1 %
30	jeasd.uomustansiriyah.edu.iq Internet Source	<1 %
31	keamanan-informasi.stei.itb.ac.id Internet Source	<1 %
32	samiam.org Internet Source	<1 %
33	Submitted to University of Northampton Student Paper	<1 %

34	Submitted to Mahidol University Student Paper	<1 %
35	Submitted to Universitas Gunadarma Student Paper	<1 %
36	jurnal.umpar.ac.id Internet Source	<1 %
37	www.ejurnal.stmik-budidarma.ac.id Internet Source	<1 %
38	id.123dok.com Internet Source	<1 %
39	repository.upi.edu Internet Source	<1 %
40	vdocument.in Internet Source	<1 %
41	lib.ui.ac.id Internet Source	<1 %
42	hdl.handle.net Internet Source	<1 %
43	sinta.unud.ac.id Internet Source	<1 %
44	www.researchgate.net Internet Source	<1 %
45	Submitted to LL DIKTI IX Turnitin Consortium Part II	<1 %

Student Paper

46	Modern Cryptography Primer, 2013. Publication	<1%
47	er.nau.edu.ua Internet Source	<1%
48	prpm.trigunadharma.ac.id Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		