**INFORME PROYECTO: CASO 3**

**Universidad de los Andes**

Facultad de Ingeniería

Departamento de Ingeniería de Sistemas

Infraestructura computacional

2023-10

**Caso 3**

Lindsay Pinto Morato – l.pintom – 202023138

Diego Alejandro González Vargas- da.gonzalezv1 – 202110240

Luis Angel Angarita Coba – la.angarita – 20XXXXXXX

03 de mayo de 2022

Bogotá D.C.

**Contenido**

[1 Pruebas por casos 3](#_Toc133926047)

[1.1 Equipo de pruebas 3](#_Toc133926048)

[1.2 SHA-256 – Datos completos de prueba 3](#_Toc133926049)

[1.3 SHA-256 – Datos de tiempo 4](#_Toc133926050)

[1.4 SHA-512– Datos completos de prueba 5](#_Toc133926051)

[1.5 SHA-512 – Datos de tiempo 6](#_Toc133926052)

[2 Gráficas 7](#_Toc133926053)

[2.1 Gráfica para 1 Thread 7](#_Toc133926054)

[2.2 Gráfica para 2 Threads 7](#_Toc133926055)

[3 Velocidad del procesador y ciclos promedio 8](#_Toc133926056)

[4 Cálculos en programa monothread: 10](#_Toc133926057)

[4.1 Contraseñas de 8 caracteres 10](#_Toc133926058)

[4.2 Contraseñas de 10 caracteres 10](#_Toc133926059)

[4.3 Contraseñas de 12 caracteres 11](#_Toc133926060)

[5 Análisis y Entendimiento del Problema 12](#_Toc133926061)

[5.1 Generación de códigos criptográficos de hash 12](#_Toc133926062)

[*5.1.1* *Algoritmos más comunes* 12](#_Toc133926063)

[*5.1.2* *Algoritmos obsoletos* 13](#_Toc133926064)

[5.2 Proceso mining 13](#_Toc133926065)

[5.3 Rainbow Tables 13](#_Toc133926066)

[*5.3.1* *Problema de seguridad* 13](#_Toc133926067)

[*5.3.2* *Relación con la sal* 13](#_Toc133926068)

[6 Referencias 14](#_Toc133926069)

# **Pruebas por casos**

## **Equipo de pruebas**

Las pruebas fueron realizadas en un equipo con las siguientes características:

|  |  |
| --- | --- |
| **Característica** | **Descripción** |
| Procesador | 11Th gen Intel® core™ i7-1165G7 2.80Ghz |
| RAM instalada | 16,0GB (15,7 GB usable) |

## **SHA-256 – Datos completos de prueba**

En la siguiente gráfica se incluyen los casos de prueba teniendo en cuenta la cadena usada, la sal y el hash generado. De igual forma se muestran los tiempos de ejecución

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Caracteres** | | | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1 Thread** | **Peor caso** | **Parámetros** | Hash: 621a13096d2107cd950ff4252924fad65bea11a417c68b65fc1c614a9d2acc18 Sal: aa Clave: z | Hash: 981b72b989c0687cc30638dc5f283e67d07633b8f50f604358a79c3f9ae8b795 Sal: aa Clave: zz | Hash: e57dabcf96354988fd33a5c0827eb8b2dbd44c837a9b5fd91690be1834fcddae Sal: aa Clave: zzz | Hash: 720d83c81396290c0e62074d4c366ab64ca181012c9709f1a77549156b686b8b Sal: aa Clave: zzzz | Hash: 4b926bb0473ef9ed07191ce5334a689372976943c0f2a161c4e0f14a4857c79f Sal: aa Clave: zzzzz | Hash: 15efe44c7db7acab75cb2cf0aa746c1ff3d95101c6492c66316615a018724015 Sal: aa Clave: zzzzzz | Hash: fc72d7e1193af0cf416f89142508689dfecdaa5bad2ddc55d7a57f4202033b51 Sal: aa Clave: zzzzzzz |
| **T** | **16 ms [0,00027 min]** | **33 ms [0,00055 min]** | **118 ms [0,00197 min]** | **809 ms [0,0135 min]** | **16177 ms [0,2696 min]** | **468930 ms [7,8155 min]** | **12221724 ms [203,6954 min]** |
| **Caso control** | **Parámetros** | Hash: 9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0 Sal: aa Clave: a | Hash: f02cb77c0369cd07cb77316bc65a7a8ae67f3cb8db030562a9e3f022fa6463ca Sal: aa Clave: ab | Hash: a720045375dc91b81d68004e3f0250777d82855d52f34d71e84038b18c1b6683 Sal: aa Clave: abc | Hash: 9b3148871f9241e2c617b0b863dfebf0dcb0a8c427f3bbd61fa98816da17895f Sal: aa Clave: abcd | Hash: ef69fb325d4eea159c59ba1fbb57dd5cf51d70f38c020c100e689a7a9bc16d4d Sal: aa Clave: abcde | Hash: 7d23cadf05bfa5ddef4c2a0869105df936ba84df898ffa01af2371d91b463cd3 Sal: aa Clave: abcdef | Hash: 55924024a1f1a306d19fee40d8deff768391b0dd162cc698b53ee7c08bb92863 Sal: aa Clave: abcdefg |
| **T** | **14 ms [0,00023 min]** | **17 ms [0,00028 min]** | **33 ms [0,00055 min]** | **126 ms [0,0021 min]** | **797 ms [0,0133 min]** | **14525 ms [0,2421 min]** | **470164 ms [7,8361 min]** |
| **2 Threads** | **Peor caso** | **Parámetros** | Hash: 4a7f3d9e08ce15c4e0670b45d345ce9626b2fc42d8f559f2977393637de008e1 Sal: bb Clave: z | Hash: f0b3ede45a21fe5ed57607713f2f2eaa685e128ddbaab84f246fa21ead45df74 Sal: bb Clave: zz | Hash: 62383eb182599634ca8dde962e95d6cabdd28c359a9bb58c187a762629557748 Sal: bb Clave: zzz | Hash: 9fd3cb4bc56dfa80611125ee572a4e75a437aa5f848bcdd95d02f3e5dda154db Sal: bb Clave: zzzz | Hash: 5a9d64400a58f7896af2fdc77e101f4a5bee419cd5ba908bbe4d2a2f648a3bb9 Sal: bb Clave: zzzzz | Hash: 8149b065fa614cc690a17ed85319f502422ccd1bd8cee0ead485c1392781d271 Sal: bb Clave: zzzzzz | Hash: 92615db1c0d4f86f3af3e6cec2079db2767ad73db5b44ad4c280d2e84c6e6dab Sal: bb Clave: zzzzzzz |
| **T** | **14 ms [0,00023 min]** | **27 ms [0,00045 min]** | **116 ms [0,00193 min]** | **652 ms [0,011 min]** | **9544 ms [0,1591 min]** | **286695 ms [4,7783 min]** | **8840251 ms [147,3375 min]** |
| **Caso control** | **Parámetros** | Hash: 715edf8ba8729420cd4d1ce85ed61954a9f531f8c548df728c407effe839296d Sal: bb Clave: a | Hash: 1d162a5c95d4698c0a3e766ae80d85994b549b877ed275803725f43dadc83bd Sal: bb Clave: ab | Hash: 7fc8889f1430b0bf873cfbea9a970d8afe4992dea4afb180cb5177711d61e8b1 Sal: bb Clave: abc | Hash: cbcd7ad0f9b011fac7d265957739d5a7466905ec6a6c4d386e99fb62cefb0a05 Sal: bb Clave: abcd | Hash: 24e53214ccad9b60a357f7777f878a2b2fecb63462215b4f2f3b5177b7bab946 Sal: bb Clave: abcde | Hash: 88aef76d58721ddf7447c0a324221ceeff031bb67f0c8a6f6279bde6a993e7d7 Sal: bb Clave: abcdef | Hash: 8de48b6cf5e37b70d24dc028bd03a2edaa9c1279e9d569fc289222ff639272b0 Sal: bb Clave: abcdefg |
| **T** | **13 ms [0,00022 min]** | **17 ms [0,00028 min]** | **27 ms [0,00045 min]** | **125 ms [0,0021 min]** | **615 ms [0,0103 min]** | **10119 ms [0,1687 min]** | **339515 ms [5,6586 min]** |

## **SHA-256 – Datos de tiempo**

Para hacer más sencilla la comprensión de los datos presentados, realizamos una tabla en la que se muestran los valores de tiempos para estas pruebas

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| En milisegundos | **Caracteres** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1 Thread** | **Peor caso** | 16 | 33 | 118 | 809 | 16177 | 468930 | 12221724 |
| **Caso control** | 14 | 17 | 33 | 126 | 797 | 14525 | 470164 |
| **2 Threads** | **Peor caso** | 14 | 27 | 116 | 652 | 9544 | 286695 | 8840251 |
| **Caso control** | 13 | 17 | 27 | 125 | 615 | 10119 | 339515 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| En minutos | **Caracteres** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1 Thread** | **Peor caso** | 0.00027 | 0.00055 | 0.00197 | 0.0135 | 0.2696 | 7.8155 | 203.6954 |
| **Caso control** | 0.00023 | 0.00028 | 0.00055 | 0.0021 | 0.0133 | 0.2421 | 7.8361 |
| **2 Threads** | **Peor caso** | 0.00023 | 0.00045 | 0.00193 | 0.011 | 0.1591 | 4.7783 | 147.3375 |
| **Caso control** | 0.00022 | 0.00028 | 0.00045 | 0.0021 | 0.0103 | 0.1687 | 5.6586 |

## **SHA-512– Datos completos de prueba**

En la siguiente gráfica se incluyen los casos de prueba teniendo en cuenta la cadena usada, la sal y el hash generado. De igual forma se muestran los tiempos de ejecución

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Caracteres** | | | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1 Thread** | **Peor caso** | **Parámetros** | Hash: bac065c32591e710dd937023d6cf6446916520d0f255a566361c715671a1be56f855387037aead2d0e9b43a76aab0dd44e380905f2e83d305a44830569245386 Sal: aa Clave: z | Hash: 53ae4a3053a25b3a5eb1191f55918a8f0bee2ccd46d2aed1f9e187768bc5d24b697b6e628ee78cb3e0e521568f1ea0b70b94c61318386069d21c71c42ff3fb30 Sal: aa Clave: zz | Hash: 6ecca1881d4ee813bc460a540b07045d29aa718b5d66fcfd7a6c49989acc9d6dc59f5c6190b8c216af171b86f58ab994cf185a121caca244689082a619ae1451 Sal: aa Clave: zzz | Hash: d990b82ffb1239e596d9c13ffac18e3b1c6502e01060a6c27cbe827f7ac7ef8e08dc3818f4dd046701f42bc492a726512038325913bced91ea6d409d2ea22d5c Sal: aa Clave: zzzz | Hash: f995bdb0774494abd6b5712e54e72d11adc96c7fef7c70bd3683b6de856147a8b9df194b4067d9381cb6030612a56b6ff55b0fdc84811dd408752fa3eea511ce Sal: aa Clave: zzzzz | Hash: 3d5f2f7fbec7e073b79812676de9b67caed32488e9d92063ece27e69eaaedc2ba442b1e2e3d7e35eaed617538abc340e03baa0c9cbcd0a25e41f108bf44ffcbf Sal: aa Clave: zzzzzz | Hash: dd1aa0720b2065b4b9729787830f54e9f3b433f213e0644a1aab3982a1e51be0efd5c0881cdac10c1331872a2a562a33e9d14d533654defe3945c2197f70d077 Sal: aa Clave: zzzzzzz |
| **T** | **17 ms [0,00028 min]** | **40 ms [0,00067 min]** | **163 ms [0,00272 min]** | **1539 ms [0,0257 min]** | **37117 ms [0,6186 min]** | **1024396 ms [17,0733 min]** | **26754741 ms [445,9124 min]** |
| **Caso control** | **Parámetros** | Hash: d6f644b19812e97b5d871658d6d3400ecd4787faeb9b8990c1e7608288664be77257104a58d033bcf1a0e0945ff06468ebe53e2dff36e248424c7273117dac09 Sal: aa Clave: a | Hash: 84805b4e230dc01e7c5eac81184ec3ccc4a88daffe59ea1d77448f40479d3ae09daeaf0b2c73a92af3a34ac633cafe0b5d4df02ea6c22260d8e070b334a8e147 Sal: aa Clave: ab | Hash: 1424f8111080f42f9ee1c18923de551dacd68591130f6485c3b338b11c7f9f752b802ef6d93211cf48d84c4a82ab127a1a389a9fa3fa6236f95f94b418e47fa3 Sal: aa Clave: abc | Hash: dd2c00fc4db949dc0cf2ff4742cb502b53ec91c73de24c3aa50f26812691fd4c89531dec369b81bc8764a61fa4a527332944d84272aeb012a35496e1ed723855 Threads: 1 Clave: abcd | Hash: f646ca90f3391c326fae4ae1749951035b8574801b4243bd32a8b7dbea1e8979d1eadf71f52c86045880bb48f0c61703a6b744aecb5d174555b3714924e13110 Sal: aa Clave: abcde | Hash: 498c1c4b29936c11acb249841f904559b99fc297897dba001042c1128be75e8fca4d144f9a02f0711c9465f02788ddc03372ed73df413e2ddc71d34ef8e9c92 Sal: aa Clave: abcdef | Hash: 66eb965197971cf8185cd8860a410aaaa20cc33a218a03e5196f64d977039656e27315ac89c5e09fd0823bdde9aac8b7e0756ba304fecaa9470491ef0c4a5235 Sal: aa Clave: abcdefg |
| **T** | **13 ms [0,00022 min]** | **15 ms [0,00025 min]** | **41 ms [0,00068 min]** | **161 ms [0,0027 min]** | **1417 ms [0,0236 min]** | **41679 ms [0,6947 min]** | **1029349 ms [17,1558 min]** |
| **2 Threads** | **Peor caso** | **Parámetros** | Hash: f7ccea4a203504b011fbfeba7ddfb8b2a117bb346dff8378aac91476d1b1c51f3600db88b837cc9877dce0fb932d0f2bf57c701b1215dddc634cfc321e09355 Sal: bb Clave: z | Hash: 8b4630fcb8ff76565b3f31e2ef4173b37a73d549c60aaa2ebd168b946117ad4accac8e7d78509797dd5427fc061be65021ea31d81e1eb59bb1cfcf8c4ffd379b Sal: bb Clave: zz | Hash: 7726ce0829600040555744f8dc5786564b0a93071d9c0935d01929f6fd22987e6d61e6cf3502d1faf5c98879dfd2aa5bb720063d4086149becdad70f70da23fc Sal: bb Clave: zzz | Hash: 7336f841340d1d234d6c749bf1a99dad25398e1a908f4ee1fc0113a18d5c3c49a1cf5d9a6c400dab80dea51cac44ab8326454fef52e926d95758e3b1a26721b3 Sal: bb Clave: zzzz | Hash: 1593b83854d1f2431d775dedbc1f15aed4e911e43453aa0466245421127b8e5fd30b2197149d7cdd0fc4f249ae3d60b83accd956e9757f5aa9068da746374a03 Sal: bb Clave: zzzzz | Hash: b2a284bbca4e911b40fdb5dc092a64011aacc82722ab405069c6213865ab91c49e795efe908b7e16ef017c20d2e85e077a4921e0ec5ef267d02bcf224a2804dd Sal: bb Clave: zzzzzz | Hash: 109330da00e967412cbea09b6df0272a9fccbb00960ad74b3b5b4cf3dfc28183c6df146ea0359ee9d81c3e23acc7710d87f182da84cb83ea269eb3f74965c310 Sal: bb Clave: zzzzzzz |
| **T** | **16 ms [0,00027 min]** | **33 ms [0,00055 min]** | **162 ms [0,0027 min]** | **1034 ms [0,0172 min]** | **18555 ms [0,3093 min]** | **662663 ms [11,0444 min]** | **19890983 ms [331,5164 min]** |
| **Caso control** | **Parámetros** | Hash: 198ec4853b4fa11a26a55c9d6ac1d06e6af9401772906c4313a0d39b31891ad07860379d699c28f3840ec26b1f3e1ce33448e095496b6299ffe8fc090d71fff6 Sal: bb Clave: a | Hash: 9ade82ae3c5d54f8375f348563a372106488adef16a74b63b5591849f740bff55ceab22e117b4b09349b860f8a644adb32a9ea542abdecb80bf625160604251 Sal: bb Clave: ab | Hash: 9d99b05ea98c386b147ff0fbdbf04e8eda84982232c5a9829f312697055e0c1fb11b7aff70692f8ea508494eda36458ac264a1406e77e3693d97d121a55435a3 Sal: bb Clave: abc | Hash: 6349c9031340ee055eca55d0363cfeb14ea92a40fc8133c5a1ac37cb665963834915be87ddb593433ad32915814e81cf218051c0bfd10b2980a49cf159f69d0a Sal: bb Clave: abcd | Hash: d644bcc785a380e9605e0b392e80f63cf3366f31b0af13c94c4c35c9a28652e6646943e87393286f79cb68f6cb0c499bbe1fcbddd9c4fda54f3ec2e68b19fe2b Sal: bb Clave: abcde | Hash: 118bb1f833fb73cca7d7de56429ad0574c3236dafbfdc82d9e772b71ce4f3c722047750dfe12282a15aba7ef87c28a13749ffd61f309d89c6beb66bbe9c26d54 Sal: bb Clave: abcdef | Hash: e8b96a4fc1a42592e2acafc413aece080f1de741d75a3af5b05d25fd3a639c56a7f8ddcafa9ba3ddf97732ea088c6ba319d8075760dc8a2c0d00c97c11be1bc6 Sal: bb Clave: abcdefg |
| **T** | **15 ms [0,00025 min]** | **16 ms [0,00027 min]** | **33 ms [0,00055 min]** | **152 ms [0,0025 min]** | **1051 ms [0,0175 min]** | **26217 ms [0,437 min]** | **838592 ms [13,9765 min]** |

## **SHA-512 – Datos de tiempo**

Para hacer más sencilla la comprensión de los datos presentados, realizamos una tabla en la que se muestran los valores de tiempos para estas pruebas

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| En milisegundos | **Caracteres** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1 Thread** | **Peor caso** | 17 | 40 | 163 | 1539 | 37117 | 1024396 | 26754741 |
| **Caso control** | 13 | 15 | 41 | 161 | 1417 | 41679 | 1029349 |
| **2 Threads** | **Peor caso** | 16 | 33 | 162 | 1034 | 18555 | 662663 | 19890983 |
| **Caso control** | 15 | 16 | 33 | 152 | 1051 | 26217 | 838592 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| En minutos | **Caracteres** | | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **1 Thread** | **Peor caso** | 0.00028 | 0.00067 | 0.00272 | 0.0257 | 0.6186 | 17.0733 | 445.9124 |
| **Caso control** | 0.00022 | 0.00025 | 0.00068 | 0.0027 | 0.0236 | 0.6947 | 17.1558 |
| **2 Threads** | **Peor caso** | 0.00027 | 0.00055 | 0.0027 | 0.0172 | 0.3093 | 11.0444 | 331.5164 |
| **Caso control** | 0.00025 | 0.00027 | 0.00055 | 0.0025 | 0.0175 | 0.437 | 13.9765 |

# **Gráficas**

## **Gráfica para 1 Thread**

A continuación, se presenta la gráfica para las pruebas en las que se utilizó un solo Thread. Se resalta que para mejor entendimiento de la gráfica se utilizó una escala logarítmica en la elaboración de esta. La sal utilizada fue “aa” en todos los casos de prueba.

## **Gráfica para 2 Threads**

A continuación, se presenta la gráfica para las pruebas en las que se utilizó Utilizaron 2 Threads. Se resalta que para mejor entendimiento de la gráfica se utilizó una escala logarítmica en la elaboración de esta. La sal utilizada fue “bb” en todos los casos de prueba.

# **Velocidad del procesador y ciclos promedio**

RTA/

Para poder dar respuesta a este enunciado se hace necesario la consideración de la velocidad estimada del computador: 2.80Ghz, convertibles en 2,8 x 2^30 ciclos/segundo

Una forma de obtener el total de ciclos es realizando una división simple del tiempo que toma cada una de las pruebas y luego calcular a partir de la velocidad del procesador el número de ciclo que se toma. Para esto, nuevamente, se tomará como punto de partida el tiempo de 1 thread para 7 caracteres(& hash “SHA-512”), como procede:

Ahora bien, para saber cuantos ciclos toma esta operación se hace necesario la multiplicación de esta cifra por la velocidad del procesador como procede:

Sin embargo, aplicando los conocimientos enseñados en el curso, se realizó una segunda aproximación (con los mismos resultados), como sigue:

Ahora se hace necesaria la consideración de las instrucciones que tiene el código, en este sentido, se debió considerar el código dentro del método “ejecutarTodo”(que es el que utiliza el programa monothread, útil para el próximo punto). Sin embargo, no se hace referencia a todo el código, puesto que solo se debe considerar una ejecución de cada uno de los ciclos. Adicionalmente, se debe de tomar en cuenta que el código realiza verificaciones para cambiar el String inicial dependiendo de la cadena en la que vaya el programa. En este sentido, se puede observar que hay 6 procedimientos diferentes dependiendo del String en el que se encuentre el barrido par arealizar la tarea de verificación. No obstante, se tomó el peor caso del procedimiento con 1 thread para el siguiente cálculo de instrucciones utilizadas:

* Verificación del primer ciclo while [Línea 59]: (1)
* Verificación del primer if [Línea 61]: Realiza 2 conversiones y 1 verificación booleana (3)
* Asignación de variables locales [Líneas 63-62]: (2)
* Segunda comprobación de While [Línea 65]: (2)
* Operación matemática [Línea 67]: (1)
* Verificación del segundo if [Línea 68]: Realiza 2 conversiones y 1 verificación booleana (3)
* Asignación de variables locales [Líneas 70-71]: Hay 3 métodos utilizados en la sentencia 69(4)
* Verificación del segundo if [Línea 72]:1 verificación booleana (1)
* Asignación de variables locales [Líneas 74]: (1)
* Ciclo for que cambia todas las letras del String de análisis por n+1 a’s para empezar el análisis de una nueva longitud de palabra [Líneas 75-78]: (máximo: 7)
* Armado del nuevo String[Línea 79]: (1)
* Método verificarHash:
  + Instanciación del método de Hash [Línea 276]: (1)
  + Acomodación del método de Hash [Línea 277-280]: (2)
  + Conversión del String en bytes y obtención del digest[Línea 283-284]: (2)
  + Cálculo del hash [Línea 287]: (1)
  + Conversión del hash digest en String[Línea 290-291]: (2)
  + Evaluación de igualdad [Línea 293]: (1)

Finalmente, se ignoran los procedimientos posteriores al encuentro del hash (prints, cálculo de tiempos, etc) tal y como lo indica el enunciado. En suma, se tienen 35 instrucciones para cada evaluación.

A partir de ahí, se puede hallar el número de instrucciones por ciclo de reloj de acuerdo con el peor caso analizado, en donde se tienen que ejecutar todas las instrucciones anteriormente calculadas, es decir, el proceso con 1 thread, algoritmo SHA-512 y la cadena de caracteres zzzzzzz: 26754741ms

Finalmente, se considera el espacio de búsqueda de la siguiente manera de acuerdo con el abecedario planteado de la siguiente manera:

Revisión de contraseñas de 7 caracteres:

Revisión de contraseñas de 6 caracteres:

Revisión de contraseñas de 5 caracteres:

Revisión de contraseñas de 4 caracteres:

Revisión de contraseñas de 3 caracteres:

Revisión de contraseñas de 2 caracteres:

Revisión de contraseñas de 1 caracter:

Para un espacio total de

Así pues, se plantea la siguiente ecuación:

Se hace labores matemáticas de despeje y se obtiene que:

Finalmente, para calcular los ciclos de una sola verificación basta con multiplicar el valor hallado por el número de instrucciones del escenario tomado en cuenta:

# **Cálculos en programa monothread:**

## **Contraseñas de 8 caracteres**

Tomando en consideración los caracteres que pueden componer cada una de las contraseñas, para cada uno de ellos se tendrá el siguiente espacio de búsqueda:

Ahora bien, se hace necesario el cálculo del número promedio de caracteres de una contraseña, para lo cual se realiza un promedio ponderado de la siguiente manera:

De esta manera, se sabe que en promedio las palabras son de 8 caracteres, por lo que se hará los cálculos con esta cifra. Que se traducen en, siguiendo el patrón del literal anterior, un espacio de búsqueda total de claves de:

Así pues, se plantea la ecuación para hallar el tiempo de ejecución:

=140,83 años

## **Contraseñas de 10 caracteres**

Tomando en consideración los caracteres que pueden componer cada una de las contraseñas, para cada uno de ellos se tendrá el siguiente espacio de búsqueda:

Ahora bien, se hace necesario el cálculo del número promedio de caracteres de una contraseña, para lo cual se realiza un promedio ponderado de la siguiente manera:

De esta manera, se sabe que en promedio las palabras son de 10 caracteres, por lo que se hará los cálculos con esta cifra. Que se traducen en, siguiendo el patrón del literal anterior, un espacio de búsqueda total de claves de:

Así pues, se plantea la ecuación para hallar el tiempo de ejecución:

=856,827261 milenios

## **Contraseñas de 12 caracteres**

Tomando en consideración los caracteres que pueden componer cada una de las contraseñas, para cada uno de ellos se tendrá el siguiente espacio de búsqueda:

Ahora bien, se hace necesario el cálculo del número promedio de caracteres de una contraseña, para lo cual se realiza un promedio ponderado de la siguiente manera:

De esta manera, se sabe que en promedio las palabras son de 12 caracteres, por lo que se hará los cálculos con esta cifra. Que se traducen en, siguiendo el patrón del literal anterior, un espacio de búsqueda total de claves de:

Así pues, se plantea la ecuación para hallar el tiempo de ejecución:

=5212,937058 millones de años

# **Análisis y Entendimiento del Problema**

## **Generación de códigos criptográficos de hash**

### ***Algoritmos más comunes***

Hoy en día existen numerosos algoritmos de hash para seguridad, sin embargo, algunos de los algoritmos de hash más comunes son:

* **SHA-1 (Secure Hash Algorithm 1):** Este algoritmo toma la salida y genera una salida de valor hash de 160 bits. Este algoritmo dejó de ser seguro en el año 2005 debido a algunos ataques exitosos a sistemas que utilizaban este tipo de algoritmos. Igualmente se demostró que era posible que con 2 entradas diferentes se generara el mismo resultado lo cual puede ser una amenaza para la integridad y la seguridad de los datos compartidos. Todavía se puede encontrar este algoritmo en algunos sistemas heredados, pero se desaconseja el uso en nuevas implementaciones.
* **SHA-256 (Secure Hash Algorithm 256-bit):** Es una de las 6 funciones utilizadas en el secure hash algorithm 2 y es uno de los más usados por su excelente relación entre el costo computacional y seguridad. Es por esto por lo que es ampliamente utilizado en la tecnología blockchain, incluyendo Bitcoin. Se utiliza para asegurar la integridad de los bloques de transacciones y proporcionar una prueba criptográfica de trabajo. Otros algoritmos de la misma familia son:
  + SHA-224
  + SHA-256
  + SHA-384
  + SHA-512
  + SHA-512/224
  + SHA-512/256
* **MD5 (Message Digest Algorithm 5):** Es un algoritmo hash criptográfico unidireccional el cual toma una cadena de cualquier longitud y la convierte en un valor hash fijo de 128 bits. Este algoritmo tuvo varias fallas, pero aun es usado. Actualmente se considera obsoleto para aplicaciones críticas de seguridad debido a sus vulnerabilidades conocidas. Aún se utiliza en algunos casos no críticos, como la verificación de integridad de archivos, es por esto por lo que aún se usa ampliamente en muchas transacciones en oficinas públicas.
* **SHA-3 (Secure Hash Algorithm 3):** Esta familia de funciones se basan en un sistema criptográfico llamado Sponge System y es una versión avanzada de la misma función hash, pero diferente a SHA1, SHA2 y MD5. Cómo su nombre lo indica, esta función absorbe a la entrada y exprime el resultado. Este algoritmo aún no ha podido utilizarse ampliamente, pero en la actualidad se utiliza en aplicaciones de seguridad, como autenticación y verificación de datos

### ***Algoritmos obsoletos***

De estos algoritmos, hoy en día se han dejado de utilizar aquellos que han demostrado vulnerabilidades criptográficas puesto que pueden permitir ataques exitosos puesto que es factible la generación de colisiones de hash para romper la integridad de los datos o encontrar debilidades en los protocolos de seguridad que confían en estos algoritmos. De la anterior lista de algoritmos, aquellos que puede decirse que se consideran obsoletos son:

* MD5, el cual ha sido ampliamente utilizado en el pasado, pero en la actualidad se considera obsoleto para aplicaciones críticas de seguridad debido a las debilidades que posee. Estas debilidades están relacionadas principalmente con colisiones que comprometen la integridad de los datos y permite falsificación.
* SHA-1 posee debilidades criptográficas por colisiones lo que indica que es posible encontrar dos mensajes diferentes con el mismo valor de hash. Se recomienda no usar este algoritmo y tratar de realizar la migración a algoritmos más seguros.

## **Proceso mining**

El proceso de “mining” o minería en bitcoin es el mecanismo a través del cual se crean nuevos bitcoins y se asegura la red mediante la verificación y confirmación de las transacciones y es un componente fundamental para el funcionamiento de esta criptomoneda. El objetivo principal es encontrar un hash que cumpla con ciertos requisitos de dificultad establecidos por la red. Se resalta que esta minería utiliza criptografía con una función de hash denominada doble SHA-256. Esto indica que un hash toma una porción de datos como entrada y la reduce a un valor de hash más pequeño. Con este tipo de hash se deben probar una gran cantidad de entradas para obtener el valor de hash.

## **Rainbow Tables**

### ***Problema de seguridad***

Las rainbow tables es una tabla creada para hallar coincidencias de un hash, es decir, contiene parejas llave, valor con contraseñas y sus correspondientes valores de hash. Es de esta manera que los atacantes pueden encontrar una contraseña invirtiendo la función de hash. La utilización de estas tablas es mucho más rápida que la utilización de fuerza bruta, pero requiere mucho almacenamiento. No obstante, es un método eficiente para descifrar contraseñas y en esto se basa su mayor problema de seguridad que es que permiten realizar ataques de manera eficiente y reducir los tiempos necesarios para descifrar una contraseña.

### ***Relación con la sal***

Los ataques con rainbow tables pueden ser una amenaza para las organizaciones porque pueden usarse para descifrar contraseñas débiles o fáciles de adivinar. Es por esto por lo que una posible estrategia para minimizar este riesgo puede ser el uso de contraseñas más complejas o el uso de técnicas como la adición de sal para que sea más difícil para los atacantes vulnerar las contraseñas. La adición de sal hace que las rainbow tables sean ineficaces porque no se puede buscar directamente en la tabla para obtener la contraseña correspondiente. Es importante señalar que los estudios revelan que las rainbow tables pueden descifrar contraseñas sin sal más rápido que con sal.

# **Referencias**

Ayala, G. (2018, julio 23). ¿Qué es SHA-256? Bit2Me Academy. https://academy.bit2me.com/sha256-algoritmo-bitcoin/

H. Kumar et al., (2013). *Rainbow table to crack password using MD5 hashing algorithm, 2013 IEEE Conference on Information & Communication Technologies*, Thuckalay, India, 2013, pp. 433-439, doi: 10.1109/CICT.2013.6558135.

Manankova O-A., Yakubova M.Z., Rakhmatullaev M.A., Baikenov A.S. (2023 2). *Simulation Of The Rainbow Attack On The Sha-256 Hash Function*. Journal of Theoretical and Applied Information Technology, 1594–1602.

Mehta, J. (2022, septiembre 9). MD5 vs SHA1 vs SHA2 vs SHA3 - compare Hashing Algorithms. SignMyCode - Blog; SignMyCode.com. https://signmycode.com/blog/md5-vs-sha1-vs-sha2-vs-sha3

Prashant, A. S. S. (2017*). A Relative Study on Bitcoin Mining*. "Imperial Journal of Interdisciplinary Research (IJIR), 1757–1761.