AntCrypt[⋆]

Proposal for the Password Hashing Competition

March 31, 2014

Markus Dürmuth, Ralf Zimmermann

Horst Görtz Institute for IT Security
Ruhr University Bochum
Universitätsstr. 150
44780 Bochum, Germany

[⋆] The name “AntCrypt” was chosen because the core of our construction resembles an anthill: in both, a huge quantity of small workers carry our tiny tasks in apparent chaos, however, in reality this “chaos” is orchestrated so that all results come together and form the final result.

1 Introduction

Arguably the biggest threat to password hashing schemes stems from GPUs, FPGAs, and ASICs, who provide enormous computing power which can speed up verification of a batch of passwords (e.g., in an offline guessing attack). Common constructions for password hashes use, at their core, two different methods to limit speed-up of verification operations.

- First, aggressively iterated constructions proportionally increase the computation times for verification on all platforms. The (well-understood) problem with constructions solely relying on iterated constructions is that they are typically quite fast when implemented on GPUs and FPGAs, as they can be parallelized very well.
- Frequent memory access (e.g, memory-hardness and similar ideas) are intended to slow down implementations on hardware basically utilizing memory bandwidth and memory latency. Large memory requirements (such as scrypt) will force the attacker to access main memory (on GPUs), while moderate memory usage (such as bcrypt) leaves the attacker with a trade-off between using a large number of registers and thus voiding memory access, or using fewer registers but accessing global memory.

One concern with bcrypt is that the size of the memory used in the computation is fixed to 4 kByte and cannot be changed, and that 4kByte is potentially not enough memory to effectively thwart efficient implementations on FPGAs. With scrypt, one concern is that the huge memory requirements are problematic if deployed on servers handling frequent login requests, and make the server susceptible to denial-of-service attacks. Another potential concern is that memory access is "relatively rare" in the sense that there is one hash function computation between two memory access operations.

In our proposal, we opt for a middle-ground, which seems to offer the best benefit of both worlds:

1. *Memory usage*: We use *moderate amounts of memory*, tunable with a parameter from 256 Bytes upwards, where a reasonable choice seems to be around 32 kBytes. We ensure very frequent access to all regions of the memory, similar to bcrypt and different from scrypt, to avoid previously mentioned potential problems.

In addition, our construction makes use of a (to the best of our knowledge novel) idea that aims to slow down implementations on GPUs and FPGAs/ASICs specifically.

2. *Control-flow divergence*: Our code will frequently branch depending on the current state (and thus ultimately on the password), to (i) avoid good parallelization on GPUs, and (ii) increase the size of implementations (and thus increase the cost and decrease the throughput) on FPGAs/ASICs.

2 The Key-Derivation Scheme

In this section, we will provide a description of our construction and comment on the design choices. A more detailed discussion will follow in Section 3.

2.1 Parameters and Main Data Structure

Unless stated otherwise, all data types are 32-bit words. As the prototype of the `PHS` function provides two cost parameters, we derive the internal parameters from them as follows:

- `state_bytes` defines the amount of memory used for the state in bytes and is defined as
$$\texttt{state\_bytes} = 2^{\texttt{m\_cost}+8}.$$
Analogously, `state_words` defines the number of 32-bit words the state contains.
- `inner_rounds` defines the number of iterations for the inner loop, iterating over all `state_words` state positions. We require a minimum of at least two inner rounds as follows:
$$\texttt{inner\_rounds} = \max(\left\lfloor \frac{\texttt{m\_cost}}{16} \right\rfloor, 2)$$
- `outer_rounds` defines the number of iterations for the outer loop. We require at least one outer round and define it as follows:
$$\texttt{outer\_rounds} = \max(\texttt{t\_cost}, 1)$$

The *primary data structure* is a memory buffer `buf` = [`prefix`, `memory`]. The `prefix` can be used as a to generate different hash values from the buffer. If not stated otherwise, we refer to `buf` as the `memory` without the `prefix`.

In the algorithm, we use two such buffers: `state` of size `state_words`+1 32-bit words, as well as a `rehash` buffer of 16+1 words. The size of `rehash` is equivalent the output length of the *primary hash function* + 1 word as a prefix. These buffers are accessed either on byte level as bytes $[0, 1, \ldots, (\texttt{state\_bytes} + 3)]$ or as words $[\{3, 2, 1, 0\}, \{7, 6, 5, 4\}, \ldots, \{\ldots, (\texttt{state\_bytes} + 3)\}]$.

2.2 Algorithm Definition

Algorithm 1 describes the basic structure of the derivation function. In the following, we will describe the main functions `init`, `update_entropy`, `update_state` and `compute_output` in more detail.

Algorithm 1 Pseudocode of AntCrypt

```
Require: t_cost > 0, m_cost > 0, outlen > 0, salt, pw,
Ensure: key
init(salt, pw)                                            {Initialize state}
for $i = 0$ to outer_rounds do
    update_entropy()                        {Distribute entropy over the state}
    # The following loop is referred to as update_state()
    for $j = 0$ to inner_rounds do
        int_update_state()                      {Waste time operating on state}
    end for
end for
compute_output()                                {Final output transformation}
```

Description of init(): The initialization function `init()` fills the empty state memory with its initial content and is implemented in the reference implementation as the function `phs_init()`. Please note this function addresses the memory byte-by-byte, not as 32-bit words.

1. The `salt` is copied to the beginning of the `state` memory. It is interpreted byte-wise, i.e., state[0] = salt[0], state[1] = salt[1], . . .
   We use a fixed size for the salt (16 bytes as suggested in the proposal), and do not see a reason for supporting variable sized salts: 16 bytes (128 bit) should offer sufficient security against appropriate attacks and we do not need to add any separator or length into the buffer when using a fixed length.
2. The password is appended to the array after the salt, also stored byte-per-byte. The maximum length of a password accepted is `state_bytes` $- 16 - 1$ bytes to leave enough space for the salt, the password and an end-identifier. The minimal state supported consists of 256 bytes. Thus, a 128 byte password as required will always be accepted.
3. A “password-end” marker `0x80` is appended directly after the password and the remaining space is filled with `0x00`.

Description of update_entropy() The function `update_entropy` (which maps to `phs_upd_entropy()` in the reference implementation) uses a hash function, hashing the entire `state`. As common hash functions have a much smaller output compared to `state` – e.g., 128 bit for MD5 or 512 bit for SHA-512 – we need to extend these constructions to adapt for the larger output size. In the implementation, we use the `rehash` buffer and its prefix to derive the new state.

Similar constructions are well-known in the cryptographic literature, and in the random oracle model it is easy to prove that the resulting function constitutes a secure hash function. Basically, we compute

$$h := H(\text{state}),$$

and then

$$s_i = H(i \parallel h)$$

and forming the next state as

$$s_0, s_1, \ldots, s_k.$$

The "intermediate" hash value h is also used in the `compute_output()` function for an additional feature.

Description of `update_state()` The function `update_state()` accesses the buffer `state` `inner_rounds` × `state_words` times and aims at wasting CPU cycles and efficiently slow down parallel computation on different platforms. In the reference implementation, this function is implemented as the function `phs_upd_state()`.

Algorithm 2 Pseudocode of `update_state()`

```
for $i = 0$ to inner_rounds do
    for $j = 0$ to state_words do
        res = (state[j] ROR i)
        tgt_addr = res % state_words
        reset idx permutation
        for $j = 0$ to #F do
            choose unused idx by evaluating res
            res = $F_{\text{idx}}$(res)
        end for
        state[tgt_addr] = state[tgt_addr] XOR res
    end for
end for
```

Algorithm 2 describes the update algorithm. We use a set of functions $F_i(x)$, where #F is the number of functions and use a calling sequence of these functions, where every function is called exactly once. After all $\#F$ functions process the data, the word at the target address is updated by using a bit-wise XOR.

Please note that the sequence is not pre-defined, but depends on the value `res` (and thus the initial value `state`[j]). Thus, in theory, all $\#F!$ sequences are possible.

The currently implemented functions (defined in `phc.h`) are given in Table 1. Please note that the functions are currently being evaluated and may be tweaked later.

Description of `compute_output()` The function `compute_output()` uses the `state` memory after the last outer round to generate the derived key material. It is implemented as `phs_gen_output()` in the reference implementation.

It consists of two steps, depending on the requested output length. If the output length is less or equal to 512 bit, i.e., the output length of SHA-512, only the first step is necessary.

```
/* integer operations */
#define F00(X) ( (X) + 0x01234567 )
#define F01(X) ( (X) * 0x89ABCDEF )

/* bit operations */
#define F02(X) ( (X) >> 3 )
#define F03(X) ( ROTR((X), 7) )
#define F04(X) ( (X) ^ 0x01234567 )
#define F05(X) ( (X) & 0xFEFEFEFE )
#define F06(X) ( (X) | 0x02020202 )

/* floating point operations */
#define F07(X) ( (uint32_t) ( 2147483648.L \
                * sin (((double) X)/1000000000.L )) )
#define F08(X) ( (uint32_t) ( 2147483648.L \
                * cos (((double) X)/1000000000.L )) )
#define F09(X) ( (uint32_t) ( 2147483648.L \
                * tan (((double) X)/5000000000.L )) )

/* 1/x: [1,2] -> [0.5, 1] (bijective) */
#define F10(X) ( (uint32_t) ( (double) ( 2 * 4294967296.L \
                * ( 1 / (1.5 + (double) X / 4294967296.L )) - 0.75 ) ) )
```

Table 1. List of the functions F_i used in `update_state()`.

First, we generate the intermediate hash, which would be generated during the next call to `update_entropy`. It basically is identical to the first step of `update_entropy()`, i.e., we apply the hash function to the entire state:

$$h := H(\text{state}).$$

Depending on the desired output length, we use up to 64 byte from h, addressing the buffer byte-wise and starting with byte 0.

In case more than 64 byte were requested, we use the `prefix` for the `state`, initialized with 1, and hash the full `state` including the `prefix` to derive a new intermediate value

$$h' := H(\text{i} \parallel \text{state}).$$

We use the same function used in `update_entropy()` to derive a new `state` from h', overwriting the previous `state` and append up to `state_bytes` bytes to the output. This procedure can be repeated up to $2^{32} - 1$ times, effectively producing more than $2^{40+\text{m\_cost}}$ bytes of key material.

This construction has another advantage: By storing the "intermediate" value h as final output, we are able to recompute the "next" state. This means that we can "resume" the computation of the state from a previously stored hash value, i.e., we can retroactively increase the hardness with respect to an increased parameter `t_cost` (cf. Section 3 for more details).

3 Design choices and remarks

Next, we comment on some of the design choices that underly our construction.

3.1 Implementation

One of our main intentions was to keep the overall structure and design as simple as possible, as this facilitates analysis and implementations. This also means we omitted some features from the implementation that are easy to add for a future (reworked) version. For the same reason we omitted most optimizations of the implementation and provide a rather straightforward implementation which is presumably easy to analyze. If selected for the second round we would provide an optimized version. The overall structure is very simple, with a clear distinction between the "cryptographically hard" step (`update_entropy()`), where we use established cryptographic primitives, and the "computationally hard" step (`update_state()`), where we are relatively free to do arbitrary computations that achieve our goals.

Some features that can easily be added (and will be added in future versions):

Parallelism There is a very easy modification to make the computation parallelizable for the honest server that computes the hash. Instead of processing each cell individually when computing the `update_state()`, we can read several (for example 16) consecutive cells, compute their output in parallel, and then write back simultaneously. (As we XOR the result on the target cell the order of writing does not matter.) This provides sufficient parallelism for the honest server, while not being advantageous for the attacker, as these parallel threads are still diverging.

Extending hardness Without further modifications, the above construction allows the legitimate server to increase the hardness of an existing hash without knowledge of the password, within certain constraints. It is necessary that the intermediate hash is stored in it entirety, i.e., the output has at least 512 bit. Furthermore, only the `t_cost` parameter can be increased (i.e., internally the `outer_rounds` parameter), the `m_cost` parameter needs to be fixed. Increasing the strength is very straightforward (and we will make code for doing so available in the near future). As the final step `compute_output()` is equivalent to the first part of the `update_entropy()` step (for an output length of 512 bit), we can simply resume the computation from this step on by first completing the second half, i.e., populating the entire `state` buffer from this value and then resuming with the iterations, adding so many iterations that the wanted iteration count is met, and finalizing with the final hashing.

3.2 Divergence and choice of the functions F_x

The specific choice of the functions F_i used in the construction depends on a number of factors, including the attacker's compute architecture. We are still evaluating different choices for these functions, so the currently selected functions are likely to change in future versions; any comments are appreciated.

Some important considerations are the following: If the functions take too long to compute, then an attacker can potentially queue them up to compute the same ones in parallel, thwarting the divergence of the threats. However, if they are too fast to evaluate, then the "overhead" imposed by the computations in the inner loop that are not part of the F_i's, e.g., computing the permutation, reduces the effectiveness of the divergence. (As computing the permutation incurs some substantial overhead, we consider using just a random sequence of indices; choosing a permutation, however, has the desirable property to rule out a number of timing side channel attacks as discussed later in this text.)

The overlap between different functions F_i, i.e., the potential to execute them in parallel, should be minimized; we attempted to achieve this by choosing functions with distinct assembler instructions, additionally ensuring that they are not easily transformable into each other. (Note the absence of the "bitwise invert" function, which can be expressed with an XOR.)

On using floating point operations We are aware that using floating point operations in such constructions is unusual, but we believe that they are helpful in minimizing the overlap, and they are also quite costly to implement on FPGAs and ASICs. We avoid rounding errors by converting back each result to an integer, thus being able to control any potential rounding error. But again, the specific choice of the F_x is still somewhat experimental, and we might opt to remove floating point instructions if they incur problems with portability.

4 Security

4.1 Cryptographic security

Our construction inherits its cryptographic strength quite directly from the security of the underlying hash function. We describe our construction using SHA-512, but it can be easily substituted with any other hash function with sufficiently large state/output size. We have selected SHA-512 as it is a widely accepted design which has proven security over several years, and implementations are easily available in common libraries. In fact, it should be straight-forward to prove (in the random oracle model) that, provided that the functions F_x are permutations, or at least behave "sufficiently random", then the overall construction behaves like a random function.

In general, constructing secure hash functions is a delicate matter, and large efforts have gone into the design of such functions. Therefore, we feel that it is mandatory to rely on well-established constructions to achieve cryptographic security instead of attempting to use home-made constructions. One of the beauties of our construction is that we separate the task of providing *cryptographic strength* from the task of *slowing down verification*, (cryptographic strength is largely realized by the re-hashing done in `update_entropy`, putting minimal requirements on `update_state` only, while the slow-down is largely realized in `update_state`).

The only thing that is required to really inherit these properties is that the applying the step `update_state` does not loose too much entropy. However, by our construction, applying the sequence of the F_x to the current state is always a permutation, as we XOR the output to the target value. (This is very similar to the well-known Feistel structure, which also always is a permutation for arbitrary round functions.) And if one application of the sequence is a permutation, than by repeating this argument, the entire function `update_state` constitutes a permutation.

4.2 Speed up

The intended use of function `update_state` is to slow down the computation of the password hash, thus this is the critical place to look for optimizations.

On CPUs On CPUs, we believe that only minor optimizations can be done. The pseudo-random nature of the order of applying the functions F_x means that there is very little (constant) structure that can be exploited for optimizations. Note that, when looking retroactively at one particular run, there will be structure that can potentially be exploited, however, as the structure changes for each application of the permuted chain of F_x's such structure needs to be detected during runtime. As the functions F_x are very short (ranging from a single assembler instruction up to a few), we believe that code for detecting and exploiting such structure would most likely slow down the execution more than it helps in speeding up.

Also note that we plan to consider other functions F_x in the future, and we hope to be able to provide a more formal argument regarding the potential speed up in future versions of this document.

On GPUs On GPUs, these random permutations will lead to a substantial amount of branch divergence, which means that the parallel executions of the hash function for a parallel brute force attempt (running for different passwords) will have divergent control flow. For "ideal" functions F_x with no overlap, no overhead outside the F_x, and ideally random selection, we would expect a slowdown equal to the number of functions, i.e., by a factor of 16. (Here "slowdown" is comparing the runtime for the case with convergent execution, e.g., all threads hashing the same password, with the runtime for divergent threads, e.g., when hashing different passwords in each thread.) In practice, there is overhead, e.g., caused by the final XOR and the computation of the permutation, and the functions have not entirely disjoint assembler instructions (e.g., we need some re-scaling of the values for the floating point instructions), so these ideal goals will likely not be met.

On FPGAs/ASICs While FPGAs and ASICs are very dangerous in terms of efficient implementation of brute-force attacks, the construction was chosen to render dedicated hardware attacks almost useless.

While many of the functions are easily mapped to hardware, floating point operations come at a high price. We analyzed the available cores for Xilinx Spartan 6, Virtex 6 and Artix 7 devices[**]. The CORDIC-core offers sine and square-root with 8 to 48 bit operands. For 32-bit operation, the minimum area is 3664 LUTs and 3588 FFs for sine (Spartan-6) and 975 LUTs and 1202 FFs for square-root (Artix-7) and has a latency of more than 32 clock cycles.

The floating-point core provides addition/subtraction, division, square-root and multiplication with configurable latency (time-area tradeoff) and may use available DSP cores, and the area consumption is heavily dependent on these configurations.

The use of more than one FPU function will significantly increase the area and latency of the generation on FPGAs. In addition, to support all possible sequences of the F_i functions, the complexity of the routing will increase dramatically: Every output needs to be routed to every other function as input. Thus, every function needs a large multiplexer, increasing the routing delay and increasing the critical path.

The second limiting factor is the memory usage, as fast memory cores are available but limited in size and number. To implement a 64 kByte state (`m_cost` = 8) will already use about 29 18k-BRAMs on Xilinx FPGAs. Thus, the memory area will become a limiting factor even with medium state sizes.

In practice, we think that using FPGAs or producing dedicated ASICs will not be the first choice for an attacker, as the construction is by design very cumbersome to implement and artificially adds latency, enforces complex routing and needs area-consuming FPU arithmetic.

4.3 Side channel attacks

Storing the password in memory The password is written to the `state` in the beginning and immediately overwritten by the output of the hash function. No copy needs to be stored beyond the initialization of the `state` memory. This should effectively prevent reading the password from memory.

(Cache) timing attacks The different functions F_x have usually, depending on the platform, different execution times. This could lead to timing attacks or to cache timing attacks. However, as we always use the same functions, just in differently permuted order, the time between memory access is constant (assuming that each operation runs in time independent of the data). In other words, the execution time between memory access is constant, thus no information is leaked. Then also the overall running time is constant, and no timing leak exists.

Other side channel attacks More involved side channel attacks, such as power consumption and electromagnetic emanation, are outside the scope of our consideration, as they depend on the specific architecture the code is running on. Also, they do not seem appropriate for the case considered, as an adversary that

---

[**] cf. Xilinx DS858 and Xilinx DS335 specification

has physical access to a machine verifying the correct password typically has easier attacks at hand.

5 Final remarks

5.1 Statements

We ensure that we have not inserted, and are not aware of, any deliberately hidden weaknesses in the scheme described above.

The scheme is and will remain available worldwide on a royalty free basis, and we are unaware of any patent or patent application that cover the use or implementation of the submitted algorithm.

6 Test vectors

In this section, we will provide several testvectors. Please note that the requested list of $2^8 \times 2^8$ password/seed combinations for meaningful cost factors will take a very long time to generate. To do this, please compile the included source code and run the program `phc_tv`.

The format of the output is similar to previously used formats:

```
$<seed>$<t_cost>:<m_cost>$<hash>
```

where the cost parameters are represented as two-digit numbers and the seed and hash are in hex-representation.

The output listed in Tables 2, 3, 4 and 5 was generated by the `phc_demo` program, which is also provided as source code.

```
Parameters: t_cost = 15, m_cost = 1
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$15:1$114495a461ae8b1500e532aa7173bb26e1a43a2e06bcee934a2d9cb714eed262afdb600fdf01677f8a300a1cd30eeb8d3f3c0bde4b11698c213e3a2a202cd75e
Output    : $0123456789abcdeffedcba9876543210$15:1$e0b0218195ff7bec008baf8784b361594ff81a33c7e0ce10e4fd976c6da53235e4fd9a75546d84657263ead1478164b4d6b6389a1dc6371705bbd342f281d51e
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$15:1$e49c68ebdc8fdc65e31a68bb7d3b241ed8c52dfa83ef539df76a3c0a877aca090568cab2273c7bf6dc6822f8ec852b96f0ea28e3b3339985de0660681a606e5b
Parameters: t_cost = 15, m_cost = 1
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$15:1$ca01d815190243e9510325124b419fb00d1929850f81c93829e0b58c8e37361c10b6572ee412d5ab0a573a35b66def3070c868c144c576acf95610fd4460e9c9
Output    : $0123456789abcdeffedcba9876543210$15:1$1467b73b0a48be9beed57e3b7388ad9d17fca2b700e86dda38522bac2b59ac26d03ffadc9074528eefcbeb31115a5f17222d2639eaf68da22f1128b2d8e2641d
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$15:1$708709296f3b2a870828b19cfcc3be3d2554da1ec21da46e35a8cde97ff418a320e9a8606a1539439623fd524c049f666f32d0c8e6d02080729f76094d0531df
Parameters: t_cost = 15, m_cost = 1
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$15:1$8729969794ca741ff9c88ef6e394a61aedb20dbe407d464dda9a68386b3fa653eb03e5818314e636719c6f65bc88981057e082a48c2190f776b91eaae0cf51e4
Output    : $0123456789abcdeffedcba9876543210$15:1$7b7d64c19623e7b9db5e96e20602d5eb6cf28334fc320c863c26c1b31d87d0da1cdb9f7950ad8be6e9c9a649928a2b459ab945f52da2b633221d0cf9affb601c
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$15:1$1b46cbf846ea594444615529a5e92bdc54661004d96df74fce379a8fa8462a4997fa248c324c3151e47ad19632bb1196481ab01e2116d2fec62c79aad22753a6
Parameters: t_cost = 15, m_cost = 1
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$15:1$7fb48bbfbb6a75fbd739b557e65699d6e594977a2e59269a529869255908749098 5726fb86df005da530a7d1f577cbfa4b3b1be2ee5984f5c03be888bb0d29c2
Output    : $0123456789abcdeffedcba9876543210$15:1$65f20de58609e3f5852e2bb7efd6ca4c59385ed6b910967224de36e4706a08592796f7a1af8d89df1847a845bb45257879426a51d7ea8d0fef6fe91ae98a0b45
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$15:1$44779eac6ea7f12860e60fbc26dff7c1ff48fea9380299fecb8992dec8400bcae20fa5acf9db22f9e42ff9ce258b2b0910a72f6eec3b00cc2b6e8f8060b4b218
Parameters: t_cost = 16, m_cost = 1
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$16:1$65e06dac84a3da8b58b69a1c742160c54037c84da975eec24f1cc848bd08362a99b8a06d056ec34257a321b4a2feab19a00c35b4c6fb2ed10e0b91f80246d8bb
Output    : $0123456789abcdeffedcba9876543210$16:1$47fec4d92c8ca64a1fca7fa7d5fd446f213f33905c8669c27c6cbfba2681e55db5faf201c5ee62b95f9696ba6241a7dd8b7e8591594fc5b35a507fe9c673d9ca
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$16:1$3773c5483a204d5a0644ccf3bfc8890d53bfa1c877bbe29553e57a3acea179fa18fd4acec582a469dbe6af340bf86993d5bdb60e07df7d9a172ee4be4764f135
Parameters: t_cost = 16, m_cost = 1
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$16:1$c784ce408ae67fd285a0ce35705dc6663b699fcbb03ed7c1eca32b702dadeaf2b16b5d4b4c9e53e336fd86801e0b0174e409a123b402f0405233dc9cb58b0938
Output    : $0123456789abcdeffedcba9876543210$16:1$25f004057a3142c24562ee526fe4b02b0dba62000433bc99f5c7bcbdfd720856d3b50d89ea96b5a864f53082b0652dcd21d2a9ef01ad093a460008c2f8d8df9b
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$16:1$ac594b516f64baf909fbd83a074fad85e3ecb2bbee0e075eb22afec99577af70c01e2ea65f770b6702d1389944d2e2b41e241b55620164a39326d508ee313090
Parameters: t_cost = 16, m_cost = 1
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$16:1$8ff883aa95752dedbf62ac6fef24dc7a4028d55478ffe739ec05b241c94948a783c01ab52f766039694e3a358a4c91d40b1f7f64f1fe4a93cf55431b5725ad5e
Output    : $0123456789abcdeffedcba9876543210$16:1$159a8b2395dc575feebf8783044483288901c95be30fd925cd5bb1a5ea7504bb7a228c24e500d70ce59a0fc076c1cfdabf2a8179189f848746dda50fdbd1b25d
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$16:1$458db1cce4e011eecd281880955acf6797586faed20ef3589a030957a08be6b05d73c434eee7a2a840901ea035299a663b0db6a8f2db585dc3687dad5878ef2f
Parameters: t_cost = 16, m_cost = 1
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$16:1$e41a8a102dae7a8ee2b8b9a37f3fdd8a230d3c02bb9092aaa5177e0a4051c50036f4426b264d39720259bd93348ac7c72b625fdb5a8df5bfb856bf1480310337
Output    : $0123456789abcdeffedcba9876543210$16:1$17585fcebc3bf5e9904a08fd771d0db19bfe74a2fc386afbaf9d1e70ffcbc116db45de5eedd907b277eeaffb9adeb86999b5babed36053bec2c01ed276bd4688
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$16:1$8773e25937d828eb2b30ca4031fb8cada72131b8861f1ae05b15d59988e5768ed1ab73540dfb2cf3c221fa577c8548e0ffd61a05da81fe23e456a343bc81c908
Parameters: t_cost = 17, m_cost = 1
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$17:1$3994d3a1f1b8e8c9f892e15cde440f87879193b66f6fb095d36e61718264e9eb5ca7e3de537aec41e89f516d63f735264304ec22fa8830134bf5f7a77303d034
Output    : $0123456789abcdeffedcba9876543210$17:1$17335f81ffa365480e8f0ebca2de0ef36380d731a23190614b38b414b57dccc93c661570dc1de44d82e4a60863e63acc04d6aac844efd650d30b4765c238b848
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$17:1$f9e6722302999e22eae3599ce48e613f857229c2d5b052323ed84c2c0f9ec138c25b4de48c4cbbae42e4df7ca551af9516bb5819b84f7e00dbf5972fdd1f2fdd
Parameters: t_cost = 17, m_cost = 1
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$17:1$394441cc7f1222521e81cf5aa2d5be04b9ab195a3747a497402e4c3d6a86fa353ff582d31e0aa22e248970f3a40637e9d3cb818c4f58fc8d4bd8a91fd23c3158
Output    : $0123456789abcdeffedcba9876543210$17:1$0620fa8bf893d08d98ab07324e6d8d17c4009d5ed6c093c3f4a1bb4b9758be52f2dc2676fe638def2679d8a3ef439ba5e3342f20e49c9eb26234e06b7a91caee
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$17:1$e8540d6326e6fa62293c0af90b86937e29b18c4edc912deef557f26b68b676baa5cacd1e1cbbbcd6a19dadb292f1c695ccdec835ab4b8082efcec49eee89977a
Parameters: t_cost = 17, m_cost = 1
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$17:1$98c5c1ef349c231841aa88a8848aae1fe302546ac8a960c169597412e2c48f42c9bf86c21f5c924e0d0e88d7b786f4ba175c3ebffd04d21e0e34596406078917
Output    : $0123456789abcdeffedcba9876543210$17:1$fea87c4a381b520e66876246202a3b8c7bfb79c79b1c15b686f76c78c3d6e2bf6a81600f78c5cae343f8d49d35b77d8d26048d4ffe8ed574f8e5ea5495f7f943
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$17:1$577962eb40181e1daac38acba37568c9ec02da6d27ebdf0b2616b859717f61238fed5fb91c99c60df7adeb549c545105f7a26c23c1d7a3af8e591e0830415bd3
Parameters: t_cost = 17, m_cost = 1
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$17:1$778be27376cc46e81676b014322c6f69ca558cf729368b34a9862d012ec8fbd480a0b3cfcaa9adf63da515e8c116ebdae87ca5831b99bca62298b38f52106452
Output    : $0123456789abcdeffedcba9876543210$17:1$c1b51162b8f5a8e815938c5b15088e94eaf9dc14108957f1068397e5a809409228548841edb769228984893602af548e1cbd807335ca93ee23ad78acf77bafcb
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$17:1$22842c832b6847e046619f54e851b989bef9a768fdcf128234cca327a38714623afcd4be226bdbf718de48fd6ffb329a0bb13cbcb1010bcb12c89c99a327b80c
```

Table 2. Sample testvectors for multiple `m_cost` = 1

```
Parameters: t_cost = 10, m_cost = 5
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$10:5$1b4199260f4b15730799253350c4a2b8e4d216ff2d820635c7a4076576cfbf458608e77060c8f399efb83fcf0e2a72fe742f6d337cebd51cc485ac5880c0bbbb
Output    : $0123456789abcdeffedcba9876543210$10:5$6ced729c917f3be9e66a081626878b947d4cb7569c212e75068b9ef7be761ac22367aafd4a1ffa60e2653a5865b40a8098bef1908872ab29329d0489ed2e87d2
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$10:5$d38f1d123371db95bb7c336d89ebc0b79c05e170ab46ef625cc4e5f054f6d70315c87f3bf8a0acfa39c35174baf93d4913f55eb5f384811b822cee95e2f2bed8
Parameters: t_cost = 10, m_cost = 5
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$10:5$111329367d4b4521145afe3dadd684567e3c2b12f28debb5e3e572aa7b272872531e92bdc6ea3fbf6e9eee391c7bc715f247e363148d59f7a016810195424624
Output    : $0123456789abcdeffedcba9876543210$10:5$2f29a614354f6cd85843ae67a7a27d5c05b2e8f26b1d6237ef3c6e06d594b04594894f15d1b5831a7ed783f19b445249f3bb0389e062a4a24ae5b7fff764ccc1
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$10:5$5dcbc80755ca543f6b706f0c93f83fcd6de25ccb6e63d719f8546a1b06da4f13dc4c628df19c2badfba298d7e0502e63bc1a7b4bfee8ef9c7dd44234e3c7ed32
Parameters: t_cost = 10, m_cost = 5
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$10:5$042c195d87f3ca12eba8211095fa8817052027f618e3e822a73bf7366cbf1d0d70a17de67b4124311be6380ec268c9bf004342231f7a8bc1fad95ac3b40ea96d
Output    : $0123456789abcdeffedcba9876543210$10:5$6b673833c20dce13c968d222a10e57a59cff9124f194efb09cf4b8ee294235f725d9fd7ab0761227d74cdf3c0324b3bcce84a58e35c1c9deda4848fbeda70899
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$10:5$fb62a3453699af45754bb2c4cbc89a7e5105ae73a41e157eb204e3f91249693e017471f871525dd5006cb1bbc7155caa8936ee1d954e2c89feed75316b9fff76
Parameters: t_cost = 10, m_cost = 5
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$10:5$25c75bbf2369fb8e2c0e0bcea4003b2eb0a9eb6796dc8d8a9b1911c78fe060b663c7282c74cf99a3871d660b9c6ae047749e70f80b4344ed7e323a22a6eeb2b7
Output    : $0123456789abcdeffedcba9876543210$10:5$d722282f7cfd3ccdcdcedde13459fbd21e65e4efcecb87c5c8b6c5ea83446b344cd2b382d40f4d622922852729e051c953bea6ab186688a83a11ef52eba4a6b5
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$10:5$2007ed4a1c2d94904c9fa923bb1a05fb96e9ef2c720efc0081ac298a4c730f7f4474e6b59012f4f970afdfb3fe9a12c7b9a74ddb797b59c8697ae0bf1d884f06
Parameters: t_cost = 11, m_cost = 5
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$11:5$8c6de4b418933bdb4a2b882d225e96ca37eb5b3acef50b5d9d4ade78fe07a5a9bd569411fad7df10eb908465bdc1370cf3aa58598c88d79d6661d9b95e42ae32
Output    : $0123456789abcdeffedcba9876543210$11:5$21f58def934627e1e19c573bb2b4f2b4b6c8df878e12f1907ea935ee194033741a416d63be405168729799d4f753379fbd4d12f1673587e61cd3db856e903e19
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$11:5$4eadfc3b00f3bb82c1226476cb71b530b891c47879c87b246df6aa3c72495f6d685f1d7c9ff2379a89e1e982c53784079083eda2ca4555be5bcc5a22749f40d3
Parameters: t_cost = 11, m_cost = 5
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$11:5$1b2a70c689f793de7a3a32ffe051733a6bc98c82fb5fb7eada538163ae3445fc1d7c012bf6176458add4f597be94c45bd3c629db8056043a7b99ebf0c25271e0
Output    : $0123456789abcdeffedcba9876543210$11:5$2be5b1dc6c292d1990cb6f646228c4ae756c1d8117aca6abdf78fd9244a00e465d29c4455a896d62ebe9f09beb0f0891a07a46d98f2ac93015dcf376cb7e7dd3
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$11:5$0c18b345eec78fa9621f1a17bdf3641bbd516ddfd670449d0c6f53a91b9b03f2cfe25393e56b69229f3b8aed19805656e235ea7d5c5b6dd899eefbd72035d263
Parameters: t_cost = 11, m_cost = 5
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$11:5$ab6d8773078abf2cd7d2e8a69e3f850268e06e9faff9df02cbf7ef299a4a0c0d4f328b20041b1b1e25282eb6e88b97f78e9f345a6117e9e94af35eef3d67371d
Output    : $0123456789abcdeffedcba9876543210$11:5$16ff85274365c3c97342c52d554dd437e96e24d1bdc9ba699fc1a8e1909b0f4cab631ece6668fa97431f3c5a00bf3a7b93bc329a6cb69b88c892692bdb2fd0cc
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$11:5$788ed5374c5552547015fda7e9e3c9283dd9cff724b7f7d6e4269ade3d665a7f65236fcde21bd815e9d592f7b68a5e88e7785b4ef7c1a9439a4f23e05ad7b46a
Parameters: t_cost = 11, m_cost = 5
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$11:5$398f7e9b8756baa493244eecffc09ff22926af56580808f0d98bf686b4d0a03de751b751e4afe8b0298db46fc9d26e5fe5b26d714b3317804d671234f58b5ea9
Output    : $0123456789abcdeffedcba9876543210$11:5$46efa3039947d2140986b87af275b4fe51809c255fb7727cb5dda236e801c69db9583472075635519b0efc09cf7e7f63c3399811081673a7307bd3bdb96ac349
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$11:5$a6e2ebe2f5fb59d6e761e56cea942333705f6149acdcfc570a904165dd6a36956b6ef553d4870701053d91ee6876f060b93300025a5ece23a75eeee9c9702c1d
Parameters: t_cost = 12, m_cost = 5
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$12:5$b2a8f5c30fa48ff210e219e431f61f6f6ae0f98aafe3e6ed4ed53f1081b7cd94622421ae366173737a0245b06cd115a86e29b398a0c0b0ffcfc7ad11be981858
Output    : $0123456789abcdeffedcba9876543210$12:5$aa3abc18c5e7e361a42e5302264e45659e22bd2966550ce74a1e1de76cc7c3863877720c24ef072e74dd22dadc77e2fba3f799dc8602b86e87fb552f1a892aca
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$12:5$bc22ee2bdaeacf50908338fbcb1eaefab7b7aa4083edc225d9857fe84661a3966c3c03659ab937a7a090fb313102f2395acf013872e71686b7b6ceddf048ef69
Parameters: t_cost = 12, m_cost = 5
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$12:5$44b2963a0d9fb862c3bfa20c4bc34e21439f7d20576d16d0047a5b6bdae681a9765434f9cf5754f807be6663446d7550ea59abd8660b0c6bef03b24ce34c9863
Output    : $0123456789abcdeffedcba9876543210$12:5$d602b526abb0f7541e49b928838bc40551e7bb5890c42d6c19ac9c5f6c250e20f741e9fd52c37c9788a6a702565ba448af22f90e389ffb430958db361fe1b9b5
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$12:5$7b46de446d1a2ae8511141a6620e4316fdde1180456d957527ab643bf1b90b64be03ff2fe61fcadb2fb2c4bc365785471045c00380c33206a3256f7e42d6def1
Parameters: t_cost = 12, m_cost = 5
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$12:5$e464bb24817858b56e4b81fd2dc2226b7abb2b024671701e3b2f6ba7c3fd9f2d0bb58d7b2e98bec29467d590e5d1f371050c700711a452bb19c195a709d9ed0c
Output    : $0123456789abcdeffedcba9876543210$12:5$7198bfc51d32abf7a8893b3ad7229d98a34efe656c62788493a688784a13fd02722ef19f3873da1cc7753084ec78f65c9a45e259bc45856c6828cedd065eddfb
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$12:5$d9589723a3745ce463d312c885efaf1503b1e84c80740804867c8a4891e9503d9faa2e28694a8b95eb2c7bedbaf4cef06ca1e873b861795e11031034f2811dca
Parameters: t_cost = 12, m_cost = 5
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$12:5$9a0c560f421f6a9d0b7201fdbe0404d980b530cd0a6266a18d971d0f7b6e3a93ffd1499f56a879329b8650010cf604003bbefd3a012c319edf3bc8a93ad6b676
Output    : $0123456789abcdeffedcba9876543210$12:5$78467f47960b90eb740b133c6f81b080300e4b492a2f1eafa0986659186c248b357fc46b61aaa10c2b3dd8b424466b7f13eb126243a7d23e5cc5bc53603a511c
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$12:5$f0e6c5b774edc3708fb3f695fa9ef12b90bda9a40779eccda62d6b0a0cac06b948fe10f448f67c636595b2390f3cb4129f89967d3402116263e4ce78e56bf672
```

Table 3. Sample testvectors for multiple `m_cost` = 5

```
Parameters: t_cost = 5, m_cost = 10
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$5:10$fa98e8feb287a607a4fc9c6a1aa05d7b13dade896bed6c12cb6e5a2e4efc815b8dd50ef08c67c72b75a6416cad0974c90a3336cbb96434c5ca2e2c70a3e0b1ff
Output    : $0123456789abcdeffedcba9876543210$5:10$32c59f2db0c2c921166d270c5766a88e0e03aa2812925260440edbd0a0c89ec9f3214e8142a4fd6996fbca178953e8011de1d01ec5bd0fd0be4629508f37a06c
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$5:10$995b3fe69ca0f11d348a3ee5a6e1277835a7c2160c2ebca0867710097d05d4134b9c5a5c43542115ecb34692342a79078cc1b80ef7cba579be9459295bd9b281
Parameters: t_cost = 5, m_cost = 10
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$5:10$23b5d7f598762cdefb555c7533c0235be0b600c5405e42f1ca75dc4e3eadb1c266b82318a6fdd4733bf75ad02466a6310ce00780939bca2f45eb37231888b55d
Output    : $0123456789abcdeffedcba9876543210$5:10$4eddb86e96a8c6d3f4eb1902b46a0c6c4d1c6f26d9fb2c4ea17deda85867f7b4a47b50d30d6adf68492de5fdf7028c7fa7508ac1dc46a12d89669ec5d82b98a9
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$5:10$d9f0b3eece105c6d6478b5980d5e5a51ae46f73f34fc0616a3bc7d4e2cd9a5d3e15af4592ef80d4061bccf815b891d394402ecb6c44fbef0859fc699b7c7ae56
Parameters: t_cost = 5, m_cost = 10
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$5:10$1475fc2bcc883deba685c15defb576ef9ff4a4f7c65170d806fd73e811d001ceabfba60e6bacc65b07a2a2d91c060620bede13790bdf1ffcc2ad7949ec0265fd
Output    : $0123456789abcdeffedcba9876543210$5:10$6a2df69d31fc1aca0723e7fa6c7729f8d97d0b9eaabcf968a917b62661c2477001df3f5e4f4eb80bf4cd4bf912033360ac606111c3e8cefe9e0330dd6e091892
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$5:10$fb7e62a0ffa01646aa3b2bb251e2628c0bb4ec0f9e219dad27342b104ed918f91f8ac6edb8ee5d81585cfd8f705b232021519719b49bd000d5144c4544ac83b8
Parameters: t_cost = 5, m_cost = 10
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$5:10$85c985ac8809c2c9214542ead3fa2eb11d2d511a08992de826b2b6cc2a10ae92cc3be55dfce72a939a2c14bac47c8b5ad79adc16531382135ce3755f599f1031
Output    : $0123456789abcdeffedcba9876543210$5:10$669bf7a93c66de25886941dd45128464138c812ba278c2c3d3eec050263ba0af7399c63db4a4d7e768418c554f3d173c90732693ba887451ae75afe06b2ec340
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$5:10$4412d77476c75a32b60b13ce6662a886eba21d0b814674824df2e1e2c043258fe71f2ba12f429b14bb6406e9f322bbfa053ce79fabaa4fc74a037572fb43bbd2
Parameters: t_cost = 6, m_cost = 10
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$6:10$e0638a6143fa612274b7d8525d255b10491c631a7f329353f43963f3dca540f8b2c3bfabee448379944bb956c7becd35be4bc796cfd656ea48da31ce4d4cd639
Output    : $0123456789abcdeffedcba9876543210$6:10$67af23812c8c59c52121821876b3a0323fbb7b7a693d84126b7bf1fb45e87d50f982e2167fdfaefbd56549bc128af254e4da42180cec4f87f31ef4b5d25b74d0
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$6:10$1591c133c6c7f33f62ae17da8c412afbd8f6c2676e24ad7ae80778ddc88429c5bdf55b67c7a2b56e6994510ebee41392477c4f162a708825e968ef99cbb14777
Parameters: t_cost = 6, m_cost = 10
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$6:10$1400efc367636700a69eeade740548aabec17bb963e9b4606dd6001d0cd31ac5b8325afb9c2cdb980d09cf7b261c4f7592d90c47319845ce5ad528ea48640bbb
Output    : $0123456789abcdeffedcba9876543210$6:10$666ef422a9d677018929454ca333239cc477d9696e055d45ff3364d6aa91ba6330ff2b3ee1ac859b864d08186417f82ade1e98c756b3acbf431687c649e83819
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$6:10$cf9e3a8e1e597ee1e122a32c81953274e3a95b2c0f20fdbec59ed6ee550ad718b0c432d2f86ee2b8df3caebb191e32a68dfcd990ae97e2711702ab652ef4d8e7
Parameters: t_cost = 6, m_cost = 10
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$6:10$4db8c549fccb47a179b73d80027c45f218edceff54530fe0aa6647bc02d8dcae03a854f44cb66f2e69b9c007ee6eecf45a97b9b6a669a42ac28bfd28528a9da1
Output    : $0123456789abcdeffedcba9876543210$6:10$01a61c37a38571ccddb86d97e8eb24ff3118dd4fb799f96d8673f4813f5fd29b15a7645acad42131ae7d4ff97e96e062608aadeaf9b71cae340687f7fefc198a
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$6:10$acc4182601ef437c7ed6300550d82f336827d0fbb9c6f088aebf79557f1b3172fe2944bc1072921c0e83866a1b32b0f6aa1c6fca41fd5accc64ff528ebd3d8e5
Parameters: t_cost = 6, m_cost = 10
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$6:10$c3e81e51125c579fceeb66567826191c824f13fcab3f2464b3503105b81d4e05d9af90bacaeea87815173aaaf9552be3fd56ae0477ab95d5114cd0433f36ae89
Output    : $0123456789abcdeffedcba9876543210$6:10$1a7ff28e76ad8f4d63888991e4745fcef9632b7ef5cbf30b7d9edac0d6ea14031a0f40ffac6478aead8151fa908bb9e14d32717a0cc215239e7ec792cf4a99ad
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$6:10$f7bc33efc3f47498c33fca3e2f1ae838a6810de3caa07c553631841e408ad720c83550dca23eafc236d99d488080c4b9a663d71204ac015662b79074cc1982ad
Parameters: t_cost = 7, m_cost = 10
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$7:10$5b9d21be7cabdb6c23b6a1cfb65ec8077e7a9b355b388168ebcef6d92a9507cb828b74d03b93e52bb7cbee3b4d51062fe75b1ce14df973583db3c75ecd8b1535
Output    : $0123456789abcdeffedcba9876543210$7:10$fd81b1c33f99b5fc4ed6f7a732dbfe031d9b737982154c9ee3de32a2043f1981983c1f5fe4dec2f9a6faf223f145a841e43f34aa94ba51bbde3ac6e87ad08ca6
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$7:10$a05bedf903233a09e892d9783479cffa28e74d7f091d4bdf17399f2032696b0896dda55f7fb58382543a4e9627e7a4f7828e39ed112d5635d984767178a533bc
Parameters: t_cost = 7, m_cost = 10
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$7:10$844c374bec1ef5917831c738b18d676c9063ed018282b357728bc74d0ef5fc53a12c5d418d31f1075682cff6db1d5de6958c36f23834bc0217a1d33d3da0b8a1
Output    : $0123456789abcdeffedcba9876543210$7:10$6710f0f208e8104b5952ff7be58eee427ab351c9cba368c4be004bd7c644da70e9a3739e00f58358fa3163703a40063911cfef65522183f82eb697691f756e86
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$7:10$458715aa62abc0767ba9f4efdcd4d5c4c9359f4caced2ce8f1bbed65ada5b240a66851578a2288c404db1c4ef6ed5707fcc8831945aaa51aaf4b16a80fc352e1
Parameters: t_cost = 7, m_cost = 10
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$7:10$935c283c476bf1a1579de4caf2696ea43095fc1f536b7d47f132dac25666d80db51bf8bbcbd9ec0ec7adb88473b69024a61eb2d09f36bb3c19f838b91859e13a
Output    : $0123456789abcdeffedcba9876543210$7:10$a8f4e8569f8231ea54897e207ba7aa9c69a524102b11b92c50127d93ea09ae5e18de5094ce097b6583db89c5efae7a3edd401668efb17a171d725bfc0378e3b7
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$7:10$0e514d8570b2eee836987e1d19f5e2e2928dab776ef490eb43e28873df3cf1a73a4d13856d13fd9c9340a9c9e90cf8facea524f96b9eda5bb7c1b951f928bd58
Parameters: t_cost = 7, m_cost = 10
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$7:10$b093a65ffa85c18a357c0216c7ecef42f7438520bde79109bbafe0b1d8a0478ff415c85d63334c6e154566418888d896bd96ee0f92ec129389f5bac24423c47c
Output    : $0123456789abcdeffedcba9876543210$7:10$760cea496193d665af62c46a1ff1eb5cb88625292dd1a2afaa58e0550cdccae596fbb235d62f1f108bc66fbee93217bcfffe8124f3cb2737b33fffa50c908f49
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$7:10$59fecba279f016f54235d60dc543c1ef3e18752c25a06b9a4607a97bb66898728c17e8fb987abee08a6ca16e2c5f5b49224c66dcd568f7787b9ea890d4ac023b
```

Table 4. Sample testvectors for multiple `m_cost` $= 10$

```
Parameters: t_cost = 1, m_cost = 14
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$1:14$4ceb63b51cd1b6bea96a7347cd9dd3b8d1e0f31d600214249ef17c872ee3a18847e59503c59fb6bca3add84b2ccc3f7625703e476e07d6690951f1c619cd9a7b
Output    : $0123456789abcdeffedcba9876543210$1:14$cf4b36b561374a45782ffaa8e7a20f52b77897317c6f6cd5a1555a5d1ecd78cce345def6c76494c857ea7f8a004018977b5238d04a3718a70f7d565e3de26504
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$1:14$0982c64f4730c29dce65c48155ff82534c4f0f90e215415f9d67a5f829cd00fca44ba84681d5f326256e7ffa43b05727b923c9b015d463f766910ccdc9044d54
Parameters: t_cost = 1, m_cost = 14
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$1:14$c80910a4ee6a249b3088f8bb63578bbca266259a985ca5c6809a26f0a60994076ee18b649371f5073eed9f9b37864204fe95dba7c12ff6423c34cce968a2c5dd
Output    : $0123456789abcdeffedcba9876543210$1:14$76b0ebf36a1025cb37ae01a813a4a4b60b3a40b09e22fd862a3b606aa006e88b04be65046b8bfcfb0b690634785ac0f99dbba032d8df0ca02a6e5ba73bd72b2f
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$1:14$cd78e1c22c4da74264f3f1c7b62a0772adedd873d40716be1cb50d5e37cfbabd3856d7dfdcae3bda8a28c6376788db1bc1e2c0a83a96b5c47ffb542d01a65b51
Parameters: t_cost = 1, m_cost = 14
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$1:14$e85355d84496d5667b836e19321ca904b502ac8e0f2e7f074276d12290ec9ded05f55de33957e53ed4343e70ae35323e5e46a60ceb81b5b45750590bc15f14b0
Output    : $0123456789abcdeffedcba9876543210$1:14$80b3e5664c8eb60f5e80ccf8385361878a38e993fa67bd524c831681727928a7756074ce67845f300c1d0f3d323bff0d2ef71a794d11c0d8043c61ff2c7954fd
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$1:14$b36107d90639e2a25d40a0073bc9df5ae8c59d2f349044b1a4b54ac738400cc3594c2198a81520d380922e214db18578ed75ab2c669d0b4e52f50319cca54562
Parameters: t_cost = 1, m_cost = 14
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$1:14$8bb49ccd9cf1f787c9d20c02316c47f5b9e033a88c77c96e38e07177084919b4435a1478f31f4ef51742d3f3fb83119e25ae90d32937f857acaa78393517a568
Output    : $0123456789abcdeffedcba9876543210$1:14$c4150fdc185b37ca1e93c6ea3253924a9b6f3fc8f82813c5c1aebe39768ee42f8fd3982714f4e2af5aec0b5870d8e47952a0f1f0a952ed48e6ad8b4c3689e1c2
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$1:14$d1c2c6e0cc44bf0c68a6ab59307740bf2e018925f78ee30630e18c291f9c716b2d8500d192ed6e054dab92813767d098f11d654cc2365303690016ec9d38293c
Parameters: t_cost = 2, m_cost = 14
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$2:14$961d9ebfbc56692a78baece2d8557d981ee482d64eb4f9fe195361d6b78d9a92ad61b300d996ed4400c0364580869eacfa46c5898f82798829c6a546aac8857c
Output    : $0123456789abcdeffedcba9876543210$2:14$e714c86f80c58ab537f400a368ba987416176a239d017fa842c15b5a438e76f77012f26b04da1f3b02e802f5d3d3cf81eb3590ac73047a3806812345137cff1d
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$2:14$40afb9ea973eb2c1ddcc03eb575d7b3a8e446ab3110706cbcfdc820d16debe7e17b1fbb2f03d8ef20eedde72c0bbdf2dd0b76d95ca37b17afe1e2c6244809bab
Parameters: t_cost = 2, m_cost = 14
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$2:14$42a9f3ef184e067ea57a275a7dda43857adcd1b5147e1df935278abfd3d2d69731cb2d81163451b9f653911e8eb273d42b48e90c88200d902544f337abb182b5
Output    : $0123456789abcdeffedcba9876543210$2:14$8fe04c842d1091226a0090e2597c4b6c14303b245039ea1b021796fa2aa27307c642e77c412d107ba11fde25061a1d20f448222bea195174792d0638cbfbea66
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$2:14$9407239a6f3522489c329a59c04706ed94a47cfdaa8cf9885ce625cd77a9d7b2368451e51c7d04fab51666c93054b4880a3c1c7a19836b9e61b121ab383b6451
Parameters: t_cost = 2, m_cost = 14
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$2:14$e4f2229e6c3017d985c1f7c488ca2e4e339c772a26e72965b22d91e1969796f291942a24f1b2fa9f0b6dce0f6f35e643330fff53dace99d05acd1979287e738a
Output    : $0123456789abcdeffedcba9876543210$2:14$bc758aabe50f40690eb1d6886d4a29cd9d10a681fb434718f148cafd25208ccd6244b5e1239d5fa2fca5858b22f33ac2f668d5c2ca07ff72c3d7c64d2a258b71
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$2:14$07d5cd41802e053cd112c2bdab2fd10598e52ec58b616e51c52e42ceafa5616b1d15fed14d24e29390ac91e37feb63f04132a4393f5f14d74703283f33c81b91
Parameters: t_cost = 2, m_cost = 14
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$2:14$096b6f4f191cb86c140102cd07e3aad414ecac49a40e223efc99e32061d4dbb100370e8660233b2c699c36e9e235d42a3dedca2da1f207210641e5a6ea3b7c93
Output    : $0123456789abcdeffedcba9876543210$2:14$b57516b2c98ee6c32cf24b1359167f3b328790263c23260d9ed373e235f6ca3869b886b34f6ec8d7ff0fa18d416ec97bfc3b0ddc6e20e438b365e50bdde44909
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$2:14$69110d775f9189b18826535e4a46639dd93a245851bf8c006816a1f736e4a6e736da3f64e677e64a3fa4ac7b19b2c748872e101cf560f0d00ebd4d86e3a9e3c8
Parameters: t_cost = 3, m_cost = 14
Input     : Teh quick brown fox jumps over the lazy d0g
Output    : $00000000000000000000000000000000$3:14$16174c893e96cfb110af584caee53bd589ef1d99c87d1ca512a6646383e01c1cefcd176f4b261d4106300b861fcdc64a9b6a92d5e06d441c262e0d7dabb31a25
Output    : $0123456789abcdeffedcba9876543210$3:14$7794ddd898c5c29c7a347216079a07e90b83953a9e8d473821545118c0ccde91d8e0b88b0b7906e38379454193192831c332995010a41717a89ea035ede7ddec
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$3:14$0bab08aaa6202226c6a7f9b7f429f48821742e7ad6c227fa329fc7a8c5ecaa5d13feee5e54cbb766d79c5be24f2bb64b25207b2cc096e1b6e3370f3c3725c428
Parameters: t_cost = 3, m_cost = 14
Input     : The quick brown fox jumps over the lazy dog.
Output    : $00000000000000000000000000000000$3:14$3435fb6a739b91edbe9c3aaba4055e0c372dfd79de473b1bf8167395ba0660ac1f53e6d959582c8533004d90f05de1dcada54d48cea54b75877af8cb5b965db3
Output    : $0123456789abcdeffedcba9876543210$3:14$8854dc201037eaae037390558e8ecc0e55bd2017b80242e2a855f0848e1c69bf1c7f6ecb4bfc841f41901d0eb37efc88a98396c44b3b509c5849979499dc1acd
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$3:14$670c55fe6b90b51c3948f5854efee8f43fc818d40647ef9d88f0270314fdda424aef4ff186c86d21d934f98ddf80e9786a458574e0453fe9ab17652b10ab6df3
Parameters: t_cost = 3, m_cost = 14
Input     : The quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$3:14$0beee5aa9a959f14122a6714c4d75e24540320a1ae6078ff49d2d36b8cb526ee88026402ab9cae25ac362047896b771469a21a191566cc7c02d68b02d06a882c
Output    : $0123456789abcdeffedcba9876543210$3:14$ac133de45f75179daa1dffd6a6e56e751deeb36d17c2e736ef9141a6157baeb90ded8c904f18566bc4b307a7bfd67842307cd1eb624e0e05e6be25e079075c80
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$3:14$8c4df609609bb4ac1c90ef1708b069e369df2e753baf0fcf482707d54353fca53ebfd2c8407bb3e2192461f0c6bb9c1c582a86a7dd9ec4269cb8f629ff83506b
Parameters: t_cost = 3, m_cost = 14
Input     : The Quick brown fox jumps over the lazy dog
Output    : $00000000000000000000000000000000$3:14$fb956fc5f9016e18bdc98c7bc55bc2c7f953ce70ba9b99b5b5da93a6d087e93a83158707cfc02262928cd0f3a816f28d3311c7c7f4973284a199edc5c0c9d84e
Output    : $0123456789abcdeffedcba9876543210$3:14$ec29f7508697679ae2034d240e7897a1cb6016e1bc35e40725265e01b087b55b3f7cca4715d86e176df3c220a36aba2d01c26033f7f700cd819fa19529032c6e
Output    : $aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$3:14$b57d6567eb16b819dd9c97f2574fe82e0095cc055a7b58fddb560a3c9e6c0ab14b1251723ce886caefe947b119edd7c2316357296e347f1e366913e42756910a
```

Table 5. Sample testvectors for multiple `m_cost` $= 14$