T52 Cryptanalysis

Author: George Lasry
Last update: March 28, 2022

Overview

This document describes the weaknesses of the various models of the T52, and how they may be exploited. Those weaknesses are the basis of attacks developed during WW2, by the Swedish FRA and by Bletchley Park, as well as by German cryptographers (as part of an assessment of the security of the devices). They can also be exploited in modern attacks, described in high-level detail in this document.

Those attacks include:

- Attacks against **messages in depth,** i.e., encrypted using the same key[1].
- Ciphertext-only **statistical attacks**, applicable only to models with regular stepping - T52a/b, T52c, and T52ca.
- **Known-plaintext attacks**.

T52 Weaknesses

This section assumes some knowledge of the functional description of the T52 family of encryption devices, described here.

The weaknesses of the T52 include:

- **Fixed wheel patterns**: The code patterns of the wheels were kept unchanged throughout WW2, with minor exceptions. Most attacks described in this document would not have been possible (or would have been much more difficult) if the patterns were changeable.
- **A limited number of unique substitution alphabets**: In theory, the 10 control bits of the encryption/decryption unit should have allowed the implementation of 2^{10} = 1024 different ways of mapping the 32 Baudot 5-bit characters, or 1024 *substitution alphabets*.

[1] Not applicable for T52d and T52e in case the Klartext (KTF) function is activated.

In practice, the number of unique substitution alphabets was smaller, and in some models, significantly smaller[2]:
    - T52a/b and T52d: 960 unique alphabets
    - T52ca and T52e: 256 unique alphabets
    - T52c: only 60 unique alphabets[3]
- **Messages in depth:** Some models had a special lever allowing the operator to return the machine to the original starting positions. As a result, and due also to weak cryptographic discipline, messages in depth (encrypted using the same key settings) were often sent. In some cases, tens of messages were sent in depth. It is often possible to extract the plaintext of the various messages using guesses and trial and error, without recovering the key.
- **Statistically biased permutation**, allowing for statistical attacks.
- **Regular stepping of the wheels** (models T52a/b, T52c, and T52ca), also allowing for statistical attacks.
- **Weak implementation of irregular stepping** (models T52d and T52e) - a subtle weakness in the stepping logic, which allows for a (modern) sophisticated attack.

Attacks on Messages in Depth

Depths of tens of messages were not unusual in wartime. Historically, messages in depth were solved as follows:

- Writing the ciphertexts one under the other, leaving blank lines for plaintext guesses.
- Making a guess regarding the plaintext of some message, usually some word or sequence expected to appear at its beginning. Under a guessed plaintext assumption, the number of plaintext symbols that can match the ciphertext symbols of the other messages is limited.
- Trying to see if some combination of possible plaintext symbols makes sense, and use it as a guess.
- As each new guess further limits the options for other messages, their plaintexts may be eventually further recovered.

We explain here how a plaintext guess for one of the messages affects other messages. At each position, the encryption process consists of a XOR addition (denoted as $\oplus$) with an unknown value Σ, and applying an unknown permutation (denoted as Π):

$C = \Pi(P \oplus \Sigma)$

[2] More details in Weierud.
[3] The T52c was further weakened by the fact that the number of bits in the ciphertext symbol always has the same parity as the number of bits in the plaintext symbol (i.e., their number modulo 2 is equal). This allows for easy placement of cribs (mode details in Weierud).

It is possible to show that[4]:

$C = \Pi(P \oplus \Sigma) = \Pi(P) \oplus \Pi(\Sigma)$

For two messages in depth, and their respective ciphertext and plaintext symbols at a given position:

$C_1 = \Pi(P_1) \oplus \Pi(\Sigma)$
$C_2 = \Pi(P_2) \oplus \Pi(\Sigma)$

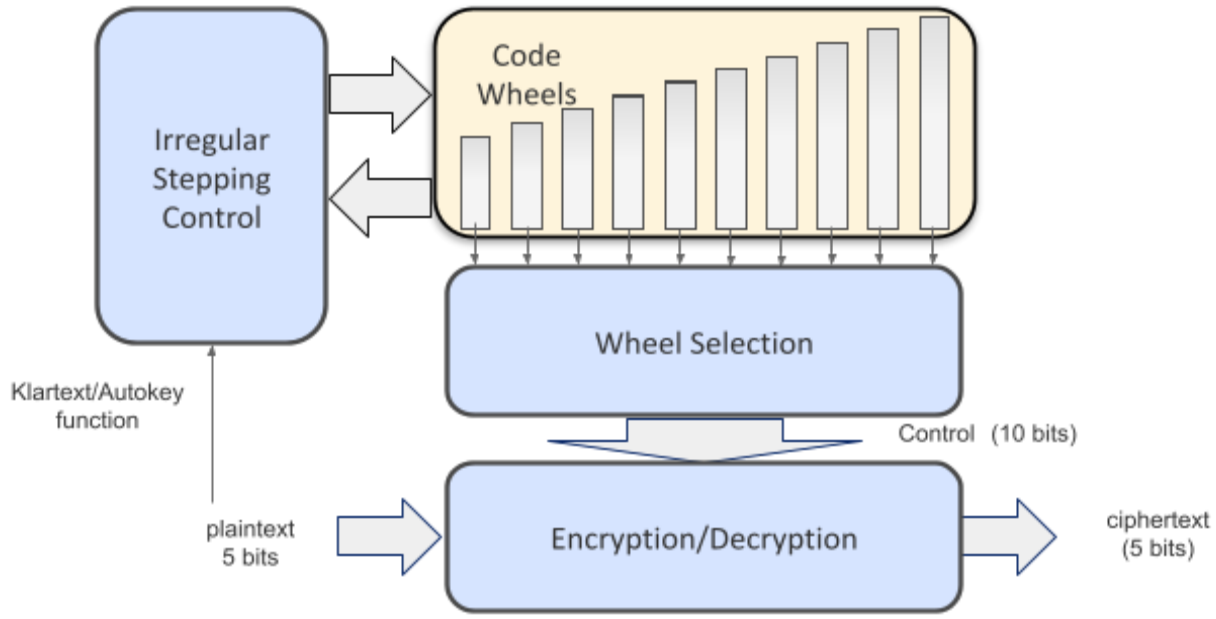
$C_1 \oplus C_2 = \Pi(P_1) \oplus \Pi(\Sigma) \oplus \Pi(P_2) \oplus \Pi(\Sigma) = \Pi(P_1) \oplus \Pi(P_2) = \Pi(P_1 \oplus P_2)$

As the permutation Π only changes the order of the bits, the number of bits with value 1 in $C_1 \oplus C_2$ is equal to the number of bits with value 1 in $P_1 \oplus P_2$. Since C_1 and C_2 are known, and P_1 is guessed, this assumption limits the number of options for P_2. The relationship described above limits P_2 to either 1, 5 or 10 options.[5]

It is possible to further reduce the number of options, with a different approach. We first present a simplified model for the T52 (see diagram below), in which:

- The wheel settings and message key unit of the T52c and T52ca models are combined in a *wheel selection unit*. For other models, the wheel selection unit simply represents the wheel setting unit. Basically, the wheel selection unit can be viewed as a switch that maps 10 inputs into 10 outputs.
- The XOR addition circuit, the transposition circuit, and the SR logic unit (for the T52c, T52ca, and T52e) are represented by an *encryption/decryption unit*, which is controlled by 10 control bits (from the wheel selection unit).

[4] See Weierud for more details.
[5] See Weierud for more details.



T52 - Simplified Design

If we consider this simplified model of T52, we can enumerate the set of 10-bit control values that when applied to the encryption unit, will map a guessed plaintext symbol P_1 to the given ciphertext symbol C_1. We then apply those 10-bit control values to decrypt C_2, obtaining all possible P_2 symbols. The number of unique P_2 options is smaller than the number of options obtained using the first method, especially for the T52c, T52ca, and T52e, which have a limited number of unique alphabets.

A different approach, also based on the simplified model of the T52, consists of assigning tentative 10-bit control values to each position, and decrypting all ciphertexts accordingly. The n-grams in the decrypted texts resulting from those tentative control values are then compared to the frequencies of the n-grams expected in the language[6], to score the decryptions. The control values can be changed iteratively to maximize the n-gram score (e.g., with hillclimbing). With enough messages in depth, this method shall reveal all or most of the plaintexts symbols.

Note that it is not necessary to test all 1024 10-bit control values, but rather only the subset of values which creates unique alphabets.

An historical attack along those lines is described in the Fish Report 68, section 3.

Statistical Ciphertext-Only Attacks

[6] Such n-gram statistics can be computed by formatting some long reference text into teleprinter format, and computing the n-gram frequencies.

Statistical attacks developed during WW2 are described in the following references:

- Fish Report 68, sections 10, 11, 12 (section 7 also relevant).
- Carlbom, Lars: *Statistisk metod för forcering av Teletypechiffer av. AB - Typ.* Link to original. Link to translation.
- TICOM Report I-45. Link (page 9 - T52a/b, page 12 - T52c).

Statistical attacks are different for the T52a/b and for models with SR logic (T52c and T52ca). We start with the outline of a statistical attack for the T52a/b.

As described before, encryption in all T52 models consists of a XOR addition followed by a permutation, $C = \Pi(P \oplus \Sigma)$. In the T52a/b, Π is driven by the state of 5 wheels, and Σ by the state of the other 5 wheels. To recover the full key (and the original plaintext), one needs to determine:

- The wheel settings - the order and function of each wheel (including the permutation switch configurations).
- The starting positions of the 5 wheels which affect the XOR addition.
- The starting positions of the 5 wheels which control the permutation.

There are 10! options for ordering the wheels, and 360 valid configurations for the permutation switches, and therefore 3,628,800 x 360 = 1,306,368,000 options for wheel settings (a little more than 10^9). There are 73 x 71 x 69 x 67 x 65 x 64 x 61 x 59 x 53 x 47 = 893,622,318,929,520,960 possible starting positions for the 10 wheels (almost 10^{18}). So the total number of options for wheel settings **and** positions for T52a/b is about 10^{27}. It is not feasible to brute-force test so many options.

In case the wheel settings are known, it might be possible to mount a hillclimbing attack, which iteratively improves the wheel positions, using the Index of Coincidence or some n-gram scoring with low n (e.g., monograms with n = 1). This approach might require a relatively long ciphertext.

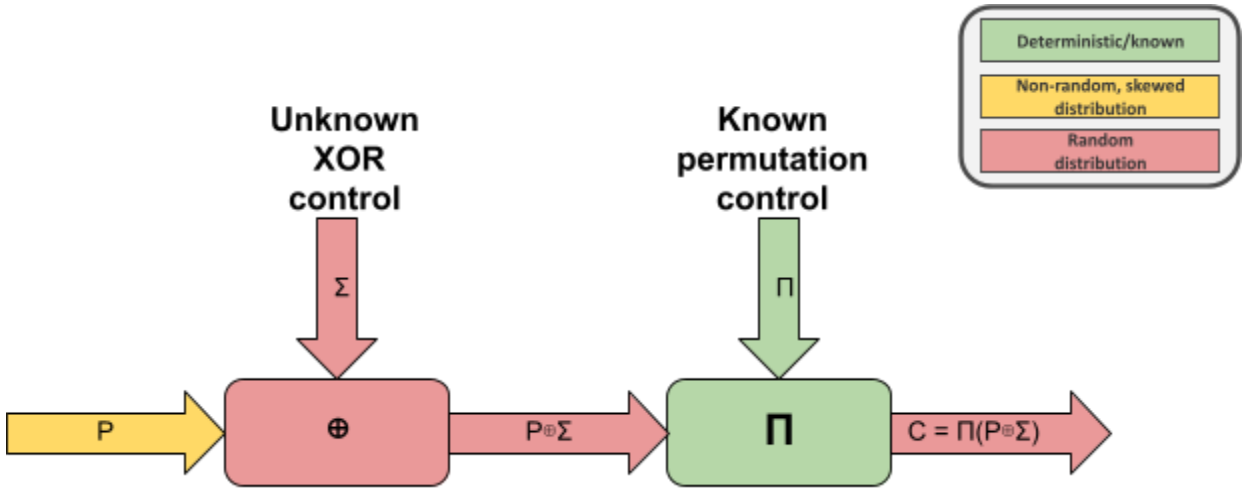
In case the wheel settings are unknown, a different approach is required, such as a divide-and-conquer attack, and either:

1. Finding first the settings and positions of the 5 wheels assigned to the XOR addition, then recovering the settings and positions of the 5 wheels assigned to permutation.
2. Finding first the settings and positions of the 5 wheels assigned to the permutation, then recovering the settings and positions of the 5 wheels assigned to XOR addition.

We first examine option (1): Any natural plain text language P is skewed statistically, as there are symbols which have a higher frequency than others, such as space (SP) or E. The XOR additive Σ, if unknown, may have any of 32 possible values, with more or less the same frequency[7]. Therefore, $P \oplus \Sigma$ is also evenly distributed. And even if we knew Π, $C = \Pi(P \oplus \Sigma)$ would

[7] With a slight preponderance of 1s.

also be evenly or randomly distributed. So determining Π based on some non-random pattern in C, the ciphertext, is not feasible. This is illustrated in the following diagram[8]:



Known Permutation and Unknown XOR

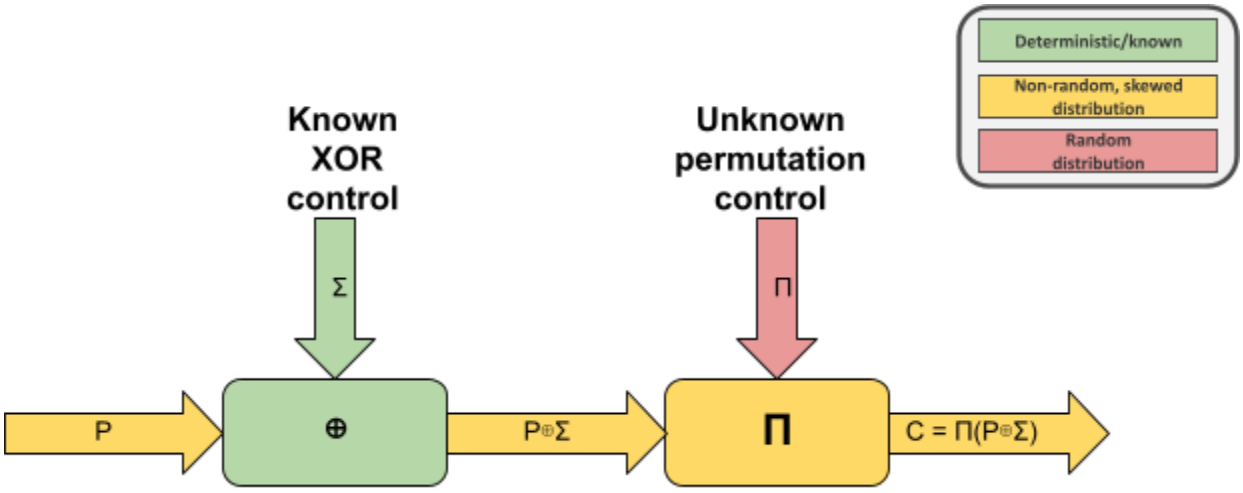
However, if we know or can guess Σ (as in option (2)), then P⊕Σ is not random anymore, and it is skewed statistically (as for P).

In addition, the permutation Π is skewed and does not randomly distribute the 5 input bits to the 5 output bits. This is illustrated with permutation switch configuration `1-2, 3-4, 5-6, 7-8,` and `9-10`[9]. If we test each of the 32 5-bit control values for Π, the distribution of the input bits after permutation is as follows[10]:

	Output bit 1	**Output bit 2**	**Output bit 3**	**Output bit 4**	**Output bit 5**
Input bit 1	9	9	2	4	8
Input bit 2	8	8	16	0	0
Input bit 3	4	4	8	16	0
Input bit 4	2	2	4	8	16
Input bit 5	9	9	2	4	8

[8] A process with random distribution applied to either a known input, or a skewed (non-random) input, results in a randomly-distributed output. Similarly, a deterministic process applied on a randomly-distributed input, produced a randomly-distributed output.
[9] Other permutation switch configuration result in a skewed permutation.
[10] The table shows the number of times each input bit is routed to a certain output bit, for all 32 possible control values for the permutation.

It can be seen that the permutation is not random. Therefore, for a known Σ, Π is also skewed (even if Π is unknown), and so is C = Π(P⊕Σ), for any known value of Σ, as illustrated in the following diagram[11]:



Known XOR and Unknown Permutation

We can use this skewed distribution of the ciphertext for a given Σ control value for a statistical attack as follows:

- Select (or guess) the 5 wheels which affect the XOR addition.
    - Guess the starting positions for those 5 wheels.
    - Divide the positions in the ciphertext into 32 classes, according to the value Σ of the state of the 5 wheels assigned to XOR at those positions.
        - For each Σ class, compute the frequencies of the ciphertext symbols at the relevant positions, and their Index of Coincidence (IC).
        - Compute the average IC for the 32 classes, i.e., the *aggregate IC*.
    - Make iterative changes in the starting positions of the 5 wheels affecting XOR, so as to maximize the aggregate IC (e.g, using hillclimbing).
- Repeat for other selections of the XOR wheels.

The process described above needs to be repeated for each of the 360 valid permutation switch configurations. When the settings and positions for the 5 wheels affecting the XOR have been recovered, the settings and positions for the remaining 5 wheels (affecting permutation) can easily be recovered, e.g., via hillclimbing.

This attack may be improved using a language-dependent scoring function, instead of the aggregate IC. If the language is known, the expected frequencies for the ciphertext symbols at positions with a given Σ value (out of 32 values) can be precomputed, by encrypting a large

[11] Even though the permutation control input is unknown and (almost) randomly distributed, the permutation itself is skewed. As its input is also skewed, its output (the ciphertext) is also skewed.

amount of plaintext in that language, and computing the frequencies of ciphertext symbols for each class/given Σ. We denote those frequencies as the *reference frequencies*.

During the attack, instead of computing the IC of the ciphertext symbols at positions with a given Σ, their frequencies are matched against the reference frequencies for that given Σ. The resulting scores for each Σ are then aggregated. This enhancement is necessary for shorter ciphertexts.

The problem is more complex for the T52c and T52ca, as the SR logic complicates the relationship between the wheels and their effect on either the XOR addition or the permutation[12]. A possible approach is to use a set of 4 wheel functions from **`1`**, **`3`**, **`5`**, **`7`**, **`9`**, **`I`**, **`II`**, **`III`**, **`IV`**, **`V`**. For each of the $2^4 = 16$ possible states, we measure the frequencies of the ciphertext symbols when those wheels are in those states. An effective set should result in a skewed distribution. An attack similar to the one described for the T52a/b can be applied, to find the 4 physical wheels assigned to those functions, as well as their starting positions.

Known-Plaintext Attacks

An historical known-plaintext attack on the T52a/b is described here:

- Fish Report 68, section 8 (section 7 also relevant).

In the same report, however, the problem of recovering the key from crib for the T52d and T52e is considered to be “hopeless.”

With modern computing, new approaches are possible. They differ for models with regular stepping (T52a/b, T52c, and T52ca) and for models with irregular stepping (T52d and T52e).

We describe first a possible backtracking attack for models with regular stepping. We initially assume that the wheel settings are known. Given a crib (or possibly, multiple cribs for multiple messages in depth), we need to recover the starting positions of all wheels. We consider again the simplified T52 model. For each pair of matching ciphertext and crib symbols, there is a limited set of 10-bit control values that when applied, the crib symbol is encrypted into the correct ciphertext symbol. For the attack, we precompute that set of valid 10-bit control values, for each crib position. Note that if there are additional messages in depth, we only keep the intersection of the sets of valid 10-bit control values, at each position[13]. The attack works as follows:

- Select one of the wheels w.
- For each possible starting position p_{start} of wheel w:
    - Set the current position $p = p_{start}$
    - For each position p_c of the crib:

[12] It is also simpler in the sense that the permutation switches are not configurable.
[13] Cribs for additional messages in depth may greatly reduce the number of steps needed for this attack. Note, however, that if the KTF/Klartext function is activated, depths are not useful, as the wheels step differently when encrypting a message, depending on the plaintext of the message.

        - Check all the (precomputed) valid 10-bit control values at that crib position, against the state of the wheel at p (active = 1, or inactive = 0), and keep only the 10-bit control values for which the relevant bit matches the state of the wheel.
        - If there are no valid 10-bit control values left, then this disqualifies the starting position p_{start} of wheel w. In this case, backtrack, and check the next p_{start}
        - Increase wheel position p.
    - Recursively repeat the whole process for the next wheel.
        - If was successful for all ten wheels, we have found a valid set of wheel positions[14].

Note that the computation time might be significantly affected by the order of the wheels tested. In general, the more options that can be filtered out earlier, the fewer steps are needed. With T52a/b, testing first the wheels assigned to XOR control will be faster. For the T52c and T52ca, several orders might need to be experimented with.

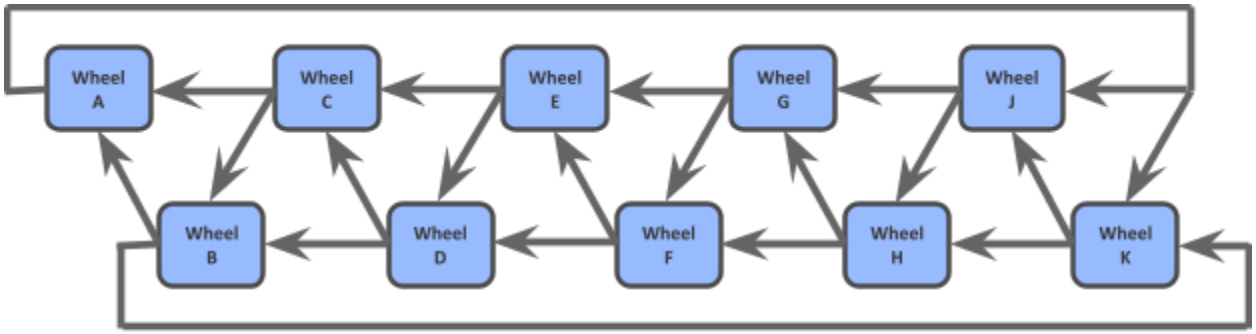
To solve the more generic problem (for T52a/b, T52c, and T52ca) when the wheel settings are unknown, there are two possible approaches:

- Repeat the attack described above for all possible wheel settings for the 10 wheels.
- Modify the attack so that it also iterates on all possible settings for current wheel w.

This attack does not work for models with irregular stepping (T52d and T52e), as in order to determine whether the wheels will step after encryption (at a certain crib position), the positions of all the wheels must be known. But this is exactly what we are trying to determine. There is a way to circumvent this obstacle, by taking advantage of a subtle weakness in the wheel stepping logic.

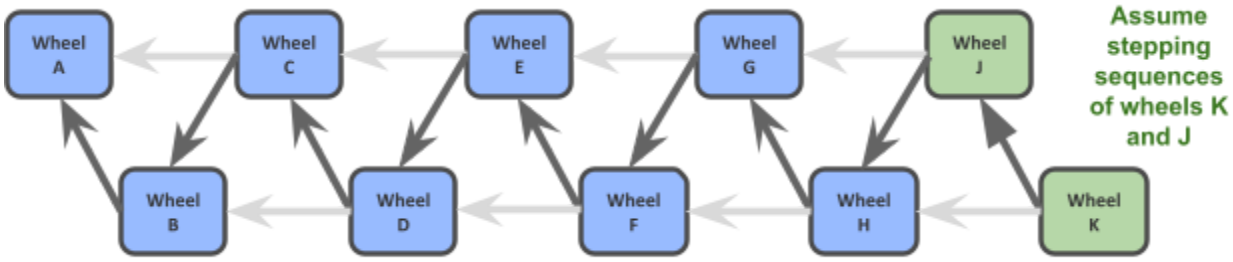
To illustrate the weakness, we show the dependencies between the wheels for stepping (for simplicity, in Klartext/KTF mode). Each wheel depends on the state of two other wheels (at certain positions). For example, the stepping of wheel F depends on the state of wheels G and H. It can be seen that the dependency is fully circular. Therefore, in theory, the stepping of wheels cannot be determined unless all positions are known.

[14] There might be more than one valid set of positions, especially if the crib is short.



Wheel Stepping Dependencies (KTF Mode)

However, it is still possible to apply a known-plaintext attack by simply testing all possible stepping sequences **for two wheels**, as shown in the following diagram:



Wheel Order for Known-Plaintext Attack on T52d/e (KTF Mode)

All possible stepping sequences for wheels K and J are tested, and an attack similar to the attack described above is carried, starting with wheel K, followed by wheel J. If the length of the crib is n=10, then $2^{n-1} \times 2^{n-1} = 2^{18} = 262{,}144$ stepping options need to be tested.

The next wheel to be tested is H, as its stepping sequence can be fully determined from the states of wheel K and J. Then wheels G, F, E, D, C, B, and A are tested[15]. Note that the dependency graph for non-KTF is different and less symmetric[16]. As for the attack on T52a/b, T52c, T52ca, the order of the wheels to be tested may significantly impact the number of steps required for an attack on T52d and T52e[17].

---

[15] It is possible to start with other pairs, such as J and H, or H and G, as long as the stepping of at least one other wheel can be determined based on the states of the wheels in the starting pair.
[16] The dependencies without KTF also allow for more possible orders, as wheels A, B, C, and D are all dependent on the same pair of two wheels.
[17] For T52d, orders that cover first most of the XOR wheels might be preferred. For T52e, several orders need to be evaluated for efficiency.